Escola Politècnica Superior de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Anàlisi de vídeo per la detecció automàtica de cares.

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Sistemes de Telecomunicació

AUTOR: Josep Mª Braup Areces

DIRECTOR: Francesc Tarrés Ruíz

DATA: 3 d'octubre de 2006

Títol: Anàlisi de vídeo per la detecció automàtica de cares.

Autor: Josep Mª Braup Areces

Director: Francesc Tarrés Ruíz

Data: 3 d'octubre de 2006

Resum

Les imatges que contenen cares són essencials per la interacció intel·ligent, basada en la visió, entre humà i computador i la recerca s'esforça en el processat de cares incloent reconeixement de cares, seguiment de cares, estimació de posició i reconeixements d'expressions. Tanmateix, molts mètodes assumeixen que les cares d'una imatge o seqüència de imatges, han estat identificades i localitzades.

Per construir un sistema automatitzat complert que analitzi la informació continguda en imatges amb cares, es necessari algoritmes de detecció de cares robustos i eficients.

La detecció de cares és el primer pas en qualsevol sistema de processat de cares i es pot assenyalar com el procés que determina si hi ha o no una cara en una imatge escollida arbitràriament, i si es present, retornar la posició de cadascuna de les cares localitzades en la imatge.

Els mètodes de detecció de cares històricament s'han classificat en quatre categories: mètodes basats en el coneixement, aproximacions a característiques no variables, mètodes de cerca de models i mètodes basats en l'aspecte.

En aquest estudi, es donaran totes les pautes per arribar a realitzar un detector de cares eficient i robust amb el mètode AdaBoost (mètode basat en l'aspecte) , proporcionant la localització exacta de les mateixes per el posterior processat.

Es donaran els criteris en l'elecció de les mostres, tant positives com negatives, per l'entrenament, de la seva adequació, del funcionament de l'algoritme d'entrenament, de com arribar a un compromís adequat entre la càrrega computacional del detector vs la velocitat de processat sense disminuir la taxa de detecció ni augmentant la taxa de falsa alarma.

I de sobretot de com escollir l'arquitectura o configuració correcta per obtenir un detector de cares en temps real, amb seqüències de imatges CIF de l'ordre de 15 fps, amb una taxa de detecció de l'orde de 94.7% i de falsa alarma de 0.00011%.

Title: Video analysis to automatic face detection

Author: Josep Mª Braup Areces

Director: Francesc Tarrés Ruíz

Date: October, 3rd 2006

Overview

Images containing faces are essential to intelligent vision-based human computer interaction, and research efforts in face processing include face recognition, face tracking, pose estimation, and expression recognition. However, many methods assume that the faces in an image or an image sequence have been identified and localized.

To build fully automated systems that analyses the information contained in face images, robust and efficient face detection algorithms are required. A first step of any face processing system is detecting the locations in images where faces are present.

Face detection can be explained as the process which determines if there is or not a face in an image chosen arbitrarily and if there is one, track the locations of each of the faces localized in the image.

Historically, face detection methods have been classified in four categories: knowledge-based methods, feature invariant approaches, template matching methods and appearance-based methods.

In this study, all the guidelines to be able to execute a face detector efficient and strong with AdaBoost method (appearance-based method) will be given, providing their exact location in order to be used in a future processing.

The criteria for the selection of samples (both positive and negative) will be given for the training, their adaptation, the training algorithm performance, how to reach a compromise between the computer load of the detector vs the processing speed without decreasing neither the detection level/rate nor increasing the false alarm one.

And above all, how to choose the architecture or right configuration to obtain a face detector in real time with image sequences CIF and about 15 fps, as well as a detection rate of approximately 94,7% and 0.00011%.

ÍNDEX

INTRODUCCIÓ

Les imatges que contenen cares són essencials per la interacció intel·ligent , basada en la visió, entre humà i computador; la recerca s'esforça en el processat de cares, incloent reconeixement de cares, seguiment de cares, estimació de posició i reconeixement d'expressions.

Tanmateix, molts mètodes assumeixen que les cares d'una imatge o seqüència de imatges, han estat identificades i localitzades. Per construir un sistema automatitzat complert que analitzi la informació continguda en imatges amb cares, es necessari algoritmes de detecció de cares robustos i eficients. La detecció de cares és el primer pas en qualsevol sistema de processat de cares.

La detecció de cares es pot assenyalar com el procés que determina si hi ha o no una cara en una imatge escollida arbitràriament, i si es present, retornar la posició de cadascuna de les cares localitzades en la imatge. Un cas simplificat de detecció és la localització que consisteix en determinar en una imatge la posició d'una única cara, sabent que la imatge només conté una cara.

Una vegada realitzada la detecció, es podrà realitzar els següents processos:

- Reconeixement o identificació de cares: on es compara una imatge d'entrada front una base de dades de imatges i es confirmarà si hi ha coincidència o no.
- Autentificació de cares: per verificar l'afirmació de la identitat d'un individu en una imatge prèvia.
- Seguiment de cares: aquest procés estima contínuament la localització i possible orientació d'una cara en una seqüència de imatges en temps real.
- Reconeixement d'expressions facials: identifica els estats d'ànim.

Les variacions associades amb la detecció de cares poden ser atribuïdes als següents factors:

- Posició. Les imatges de cares varien degut a la posició relativa entre camera i cara (frontal, rotació de 45º, perfil), i algunes característiques facials tal com ulls o nas, poden quedar parcial o totalment amagades.
- Presència o absència de components estructurals. Característiques facials tal com barbes, bigotis i ulleres, poden o no estar presents i hi ha molta variabilitat entre aquests components, incloent-hi forma, color i grandària.
- Expressions facials. L'aparença de les cares es veuen directament afectades per l'expressió facial d'una persona.
- Oclusions. Les cares poden tenir oclusions parcials per altres objectes. En una imatge d'un grup de persones, algunes cares poden tenir oclusions per part d'altres cares.
- Orientació de la imatge. Les imatges varien directament per diferents rotacions depenent dels eixos òptics de les cameres.

- Condiciones de la imatge. La imatge es crea, factors com la il·luminació (espectre, distribució de la font i intensitat) i les característiques de la camera (resposta dels sensors, lents) afecten l'aspecte de una cara.

Hi ha diferents paràmetres que es tenen en compte per avaluar els algoritmes utilitzats en els sistemes de detecció de cares. Normalment, es compara el rendiment dels mateixos mitjançant les taxes de detecció i de falsa alarma. En general, en els detectors poden haver dos tipus d'errors: fals negatiu (quan cares que deurien haver estat detectades no ho són degut a baixes taxes de detecció) i falsos negatius (una regió de la imatge es declarada com cara però realment no ho és pas, degut a altes taxes de falsa alarma).

Els mètodes de detecció de cares històricament s'han classificat en quatre categories:

- Mètodes basats en el coneixement: aquests mètodes es basen en regles que codifiquen el coneixement humà de com es constitueix una cara típica. S'utilitzen normalment per la localització de cares.
- Aproximacions a característiques no variables: aquests algoritmes tenen com objectiu trobar característiques estructurals que romanen constants per variacions de posa, punt de vista o condicions lumíniques variables per poder localitzar les cares. Aquests mètodes son designats normalment per localització de cares. Els sistemes més utilitzats són: *Facial Features*, *Texture*, *Skin Color* o *Multiple Features*.
- Mètodes de cerca de models: diversos patrons estàndards d'una cara són emmagatzemats per descriure les cares com una part sencera o com característiques facials. Les correlacions entre una imatge d'entrada i els patrons emmagatzemats son usats per la detecció. Aquests mètodes poden ser emprats tant per localització o per detecció de cares. Les propostes més utilitzades són: *Predefined Face Templates* i *Deformable Templates*
- Mètodes basats en l'aspecte: en contrast amb la cerca de models, els models o plantilles són generats des d'una col·lecció de imatges d'entrenament les quals deuen capturar les variacions representatives de l'aspecte facial. Aquests són usats per detecció de cares. Alguns dels mètodes basats en l'aspecte són: *Eigenface*, *Disribution-based*, *Neural Network*, S*upport Vector Machine*, *Naive Bayes Classifier* i *Hidden Markov Model*.

Aquests últims mètodes basats en l'aspecte, són els que millor resultat han proporcionat fins ara, ja que en funció de la variabilitat de la col·lecció de imatges o mostres amb les que es realitzarà l'entrenament s'obtindran detectors amb altes taxes de detecció i baixes taxes de falsa alarma. A més d'una gran robustes, eficiència en el sistema de detecció i reducció del cost computacional.

Dins d'aquests mètodes basats en l'aspecte, en el quals hi ha un procés d'entrenament previ per obtenir models basats en les imatges d'entrenament,

Yoar Freund i Robert E. Schapire (veure [1]), van presentar, en 1995, un treball teóric on s'utilitzava per primera vegada el mètode de Boosting. Aquest mètode permet combinar d'una manera efectiva un conjunt de regles molt senzilles que individualment no aconseguien altes taxes de detecció però mitjançant la combinació d'algunes d'elles, es pot obtenir un sistema de detecció altament potent. Aquestes regles, són un conjunt de sumes i restes dels valors dels píxels contiguts en certes àrees d'una imatge i es poden realitzar en diferents localitzacions dins de la mateixa imatge.

Inicialment es generen totes les regles possibles i cadascuna d'aquestes regles s'avalua damunt de tota la col·lecció de imatges d'entrenament i en funció del resultat obtingut es decideix quines són les que millor classifiquen la imatge candidata com objecte o no.

En aquest treball, van introduir un concepte que fou clau pel desenvolupament de posteriors sistemes, van millorar el procés de combinació de les regles fent l'elecció més eficient.

El procés és el següent: durant l'entrenament del sistema, a cada exemple l'hi es assignat un pes, un pes major a aquells exemples que són més difícils de classificar o etiquetar i un pes menor als exemples que menor dificultat presenten per la seva classificació. Aquest fet fa que el algoritme d'entrenament es centri en els exemples més complicats de classificar i així realitzar les combinacions més eficientment, per així, obtenir molt bons resultats en el detector final.

Posteriorment milloraren la seva proposta inicial presentant el mètode Adaptative Threshold més conegut com AdaBoost. Aquest sistema augmentava la potència de l'algoritme inicial de Boosting simplement amb la variació del llindar, utilitzat per designar si una imatge es candidata a contenir l'objecte a detectar o no, de cada regla senzilla avaluada sobre tot el conjunt de mostres utilitzades per l'entrenament. Amb aquesta variació es podia afinar encara més el procés d'entrenament dotant de major eficiència i robustes el detector final.

Ja en l'any 2001, van ser Paul Viola i Michael Jones en el seus treballs (veure [3] i [4]) els que varen traslladar l'AdaBoost a un sistema de detecció d'objectes en temps real obtenint molt bons resultats en comparació amb propostes anteriors. Com a regles senzilles utilitzades en l'entrenament del sistema van utilitzar les funcions de Haar, aquestes característiques són molt senzilles d'avaluar i mitjançant l'ús d'una nova representació de la imatge, anomenada Imatge Integral, es reduïa el cost computacional del sistema de detecció, també anomenat classificador, notar que un classificador es una combinació de regles senzilles, en principi les que millor poden indicar si una imatge candidata pot contenir l'objecte o no.

Tanmateix, van proposar l'ús de cascades de detectors ja que un únic detector o classificador amb una alta taxa de detecció i baixa falsa alarma no podia funcionar en temps real per el seu cost computacional. En aquesta cascada de classificadors, les primeres etapes utilitzaven molt poques regles senzilles i tenien taxes de detecció de l'ordre del 100% i de falsa alarma del 20-40%

descartant la majoria de candidats negatius; augmentant la càrrega computacional en les etapes posteriors que es centraven en el exemples més difícils de classificar.

Van tenir molt bons resultats amb aquesta configuració, podien processar seqüències de imatges de 384x288 fins a 15 fps, on es podia detectar cares de com a mínim 36x36 píxels.

Posteriorment i basant-se en el treball de P. Viola i M. Jones, Rainer Lienhart i Mochen Maydt de Intel Labs (veure [6]), van millorar el sistema de distribució de pesos en el procés d'entrenament rebaixant la càrrega computacional i una probabilitat de detecció major amb uns valors de velocitat similars. També van augmentar la col·lecció de regles o característiques senzilles utilitzades mitjançant la rotació de part de les regles usades en el treball de P. Viola i M. Jones i l'ús d'altres de noves.

A més van idear una altra configuració basada en la cascada de classificadors anomenada arbre de classificació dotant de major flexibilitat i robustes al sistema de detecció de cares mitjançant AdaBoost. Van tenir bons resultats podent processar seqüències de imatges CIF (320x240) de fins a 5 fps, on es podien detectar cares de com a mínim 24x24 píxels. Aquest treball fou inclòs en el projecte Intel ® Open Source Computer Vision Library.

Per els resultats obtinguts per R. Lienhart i els estudis anteriors amb AdaBoost, aquesta aplicació és la que s'ha usat en tot l'estudi per realitzar el detector.

En aquest estudi es donaran els criteris en l'elecció de les mostres, tant positives com negatives, per l'entrenament, de la seva adequació, del número de mostres necessàries per reduir el temps d'entrenament, del funcionament de l'algoritme d'entrenament, i sobretot, de com arribar a un compromís adequat entre la càrrega computacional del detector vs la velocitat de processat sense disminuir la taxa de detecció ni augmentar la taxa de falsa alarma per realitzar un detector de cares eficient i robust.

I de sobretot de com escollir l'arquitectura o configuració correcta per obtenir un detector de cares en temps real, amb seqüències de imatges CIF de l'ordre de 15 fps, amb una taxa de detecció de l'orde de 94.7% i de falsa alarma de 0.00011%.

CAPÍTOL 1. BOOSTING

1.1. El concepte de Boosting

El Boosting es pot entendre com un mètode que permet combinar de forma efectiva un conjunt de característiques simples que per si mateixes no aconseguirien un percentatge de detecció acceptable però que combinades eficientment produeixen un sistema de detecció altament robust i fiable.

L'aplicació d'aquest mètode passa primer per generar tot el conjunt de possible característiques simples, s'utilitza l'algoritme que anomenarem "Weak Learn". El mètode o algoritme de Boosting s'encarregarà de provar cadascuna d'aquestes característiques damunt un conjunt de imatges d'exemple de l'objecte que tindrem que haver-l'hi subministrat prèviament i que anomenarem mostres positives, així com damunt d'un altre conjunt de imatges que no contindran l'objecte i que anomenarem mostres negatives. El mètode s'encarregarà d'escollir les millors característiques tenint en compte el resultat final, és a dir, el Boosting ha d'avaluar totes les característiques generades per l'algoritme "Weak Learn" en cadascuna de les mostres proporcionades. Ens quedarem amb les característiques que menor error produeixin, que són les que millor s'adapten a les característiques de l'objecte a detectar. El conjunt de les millors característiques, s'utilitzarà per construir el detector d'objectes de manera que aconseguim una bona taxa de detecció.

El procés del mètode de Boosting es podria simplificar en dos passos: primer s'ha de trobar un mètode que ens proporcioni aquestes característiques bàsiques en funció de tots els exemples mostra, i segon, deurem combinar totes les millors característiques en una de sola.

En referència al primer punt, un pas previ i molt important, es com determinar que exemples son els més idonis per a que es puguin trobar les millors característiques simples. S'ha de tenir en compte que dintre de totes les mostres positives proporcionades no tots els objectes tindran les mateixes característiques (enfocament, rotació, il·luminació) per tant una mateixa característica no tindrà el mateix comportament per tots els exemples. Més concretament, hi haurà exemples que siguin més fàcils de classificar que uns altres. Aquest fet, ens porta a que es podrà etiquetar cada exemple en funció de la dificultat per la seva classificació.

En 1995, Y.Freund i R.E. Schapire, van proposar de dotar d'un pes major a aquells exemples que eren més difícils d'encaixar amb les característiques simples. D'aquesta manera es força que l'aprenentatge es centri en aquestes imatges més que en les imatges que sempre encaixen bé. Aquest fet, farà que el resultat final sigui robust i capaç de generalitzar davant diferents varietats de visualització de l'objecte, ja que pel contrari, intentar generalitzar més l'algoritme per què reconegui exemples extrems (molt difícil) suposarà un major número de característiques, un cost computacional major per l'entrenament del sistema i amb molta probabilitat una taxa major de falsa alarma.

1.2. Les regles simples

Com a millora del mètode de Boosting, P. Viola i M. Jones van introduir en les seves investigacions l'ús com a regles simples d'un tipus de característiques de la família de les funcions de Haar, les quals ja havien estat utilitzades per Papageorgiou en el seu estudi sobre el marc general de la detecció d'objectes, (veure [2]). Aquestes regles, són un conjunt de sumes i restes dels valors dels píxels continguts en certes àrees rectangulars verticals d'una imatge i es poden realitzar en diferents localitzacions dins de la mateixa imatge. També anomenades característiques simples o dèbils.

Posteriorment, R. Lienhart va afegir altres tipus de característiques de Haar on es realitzaven altres combinacions de sumes i restes diferents a les proposades per P. Viola i M. Jones. A més també va utilitzar àrees rectangulars rotades. Un exemple d'aquests rectangles verticals i rotats utilitzats per R. Lienhart els podem veure en la figura 1.1.

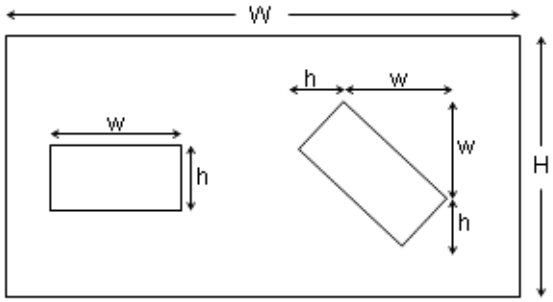


Fig. 1.1 Exemple d'un rectangle vertical i d'un rotat.

Les característiques venen determinades pel sumatori de les sumes dels píxels continguts en N àrees rectangulars :

$$feature_I = \sum_{i \in I = \{1,\ldots,N\}} \omega_i \cdot Sum\,\mathrm{Rec}\,c(r_i) \qquad \textbf{(1.1)}$$

On ω_i Є R són les ponderacions, r_i les àrees rectangulars i N de valor arbitràriament escollit.

Una àrea rectangular s'especifica com r = ((x,y),(w,h),α) amb 0≤x, x+w≤W, 0≤y, y+h≤H, x,y≥0, w,h>0, on (x,y) és el punt inicial del rectangle, (w,h) la grandària i l'alçada del rectangle respectivament, W la grandària de la imatge d'entrenament, H l'alçada de la imatge d'entrenament, la rotació αЄ {0º,45º} i SumRec(r) la suma dels píxels continguts en el rectangle r.

Degut que aquesta col·lecció podria se infinitament llarga, per raons pràctiques es redueixen de la següent manera:

1) Només es faran servir combinacions ponderades de les sumes de píxels de dos rectangles, per exemple N = 2, que es realment l'utilitzada en aquest treball.
2) Les ponderacions o pesos tenen signe oposat i són utilitzats per compensar les diferències entre la mida dels rectangles. Per evitar coincidències s'utilitza: $-\omega_0 \cdot \text{Area}(r_0) = \omega_1 \cdot \text{Area}(r_1)$. Sense restriccions s'utilitza que $\omega_0 = -1$ i $\omega_1 = \text{Àrea}(r_0)/ \text{Àrea}(r_1)$. On r_0 i r_1 són dos rectangles.
3) Les característiques imiten el camí visual humà tal com centres envoltats i respostes direccionals.

Aquestes restriccions proporciones 14 prototips de característiques, veure figura 1.2.

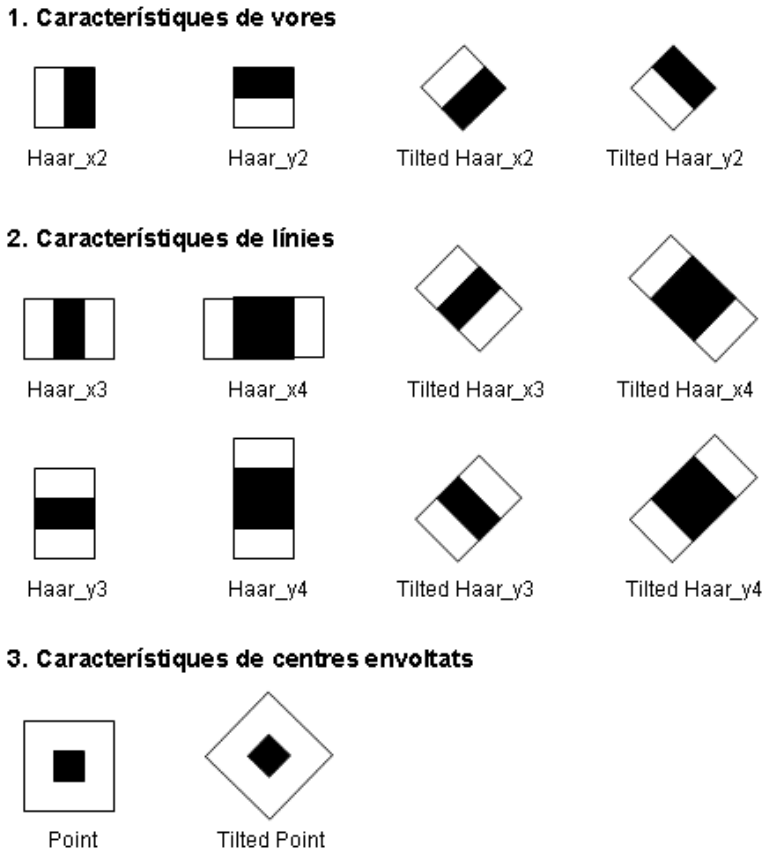


Fig. 1.2 Prototips de les característiques de Haar utilitzades per R. Lienhart.

Els prototips són escalats independentment en direcció vertical i horitzontal per generar una complerta i variada col·lecció de característiques. S'ha d'assenyalar que r_0 correspon a l'àrea total de la característica (incloent els rectangles negres i blancs) i r_1 correspon només a l'àrea del rectangle negre.

Un exemple, per avaluar una característica del tipus Haar_x3, amb alçada de 2 i de amplada 6, en el punt inicial (5,3) es representaria de la següent manera: $feature_l = -1 \cdot SumRec(5,3,6,2,0º) + 3 \cdot SumRec(7,3,2,2,0º)$.

El valor obtingut al realitzar el càlcul el compararem amb un llindar, que s'haurà escollit al provar la característica simple sobre tot el conjunt de les mostres. Si l'avaluació supera el llindar, afirmarem que no es tracta d'un objecte i si quedes per sota es consideraria una detecció afirmativa.

S'ha de fer notar que P. Viola i M. Jones només van fer servir en el seu treball les característiques: Haar_x2, Haar_y2, Haar_x3 i un altre tipus de 4 rectangles que no s'inclou en aquest treball donada la seva càrrega computacional. Va ésser R. Lienhart qui va proposar l'ús de les característiques rotades, i a més va afegir altres tipus de característiques rectangulars verticals com: Haar_x4, Haar_y3, Haar_y4 i point.

Les característiques són molt fàcils de calcular però malgrat tot, suposen un cost computacional elevat i això és un inconvenient per al sistema ja que ens retardaria enormement l'execució del detector sobre cada imatge. Com es veurà més endavant, per avaluar una imatge, el detector la dividirà en subfinestres que serà on s'avaluaran les característiques escollides i a més a més, aquestes subfinestres es calcularan també per altres escales, per tant a més de ser simples, es deuen poder calcular amb el mínim número d'operacions possibles. En aquest sentit té un paper molt important una forma de representació de la imatge denominada Imatge Integral.

1.3. La Imatge Integral

La imatge integral és una forma de representar una imatge per aconseguir una avaluació ràpida de les característiques simples. Va ésser introduïda per P. Viola i M. Jones en el seu sistema de detecció que va passar de treballar directament amb les intensitats dels píxels del sistema. R. Lienhart va afegir el càlcul per les característiques rotades.

La imatge integral permet avaluar les característiques simples eficientment, el seu càlcul és molt senzill ja que requereix un número molt reduït d'operacions per píxel. Qualsevol de les característiques detallades en l'apartat anterior, pot ser avaluada a qualsevol escala en temps constant. Aquest fet proporciona una reducció considerable del cost computacional en el càlcul de les característiques.

S'utilitza una imatge auxiliar que anomenarem SAT (Summed Area Table). SAT(x,y) es defineix com la suma de tots aquells píxels que quedin per damunt i a l'esquerra de punt (x,y). Notar que el punt en qüestió també està inclòs, veure figura 1.3.a.

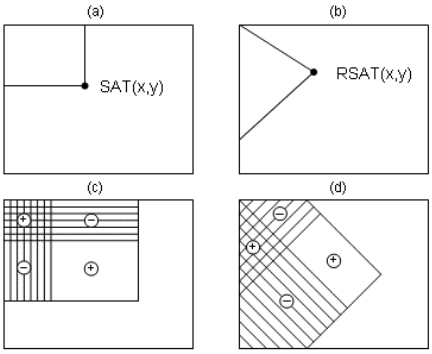


Fig. 1.3 (a) Summed Area Table vertical (SAT) i (b) Rotated Summend Area Table (RSAT); esquema de càlcul de la suma de píxels de rectangles verticals (c) i rotats (d)

A continuació veurem la manera d'obtenir-la; la SAT (x,y) d'una imatge està continguda formalment:

$$SAT(x,y) = \sum_{x' \le x, y' \le y} I(x', y') \qquad \textbf{(1.2)}$$

Pot ser calculada amb una passada sobre tots els píxels d'esquerra a dreta i de dalt a baix:

$$SAT(x,y) = SAT(x,y-1) + I(x,y) - SAT(x-1,y-1) \qquad \textbf{(1.3)}$$

on SAT(-1,y) = SAT(x,-1) = 0

La suma del rectangle vertical r = ((x,y),(w,h),0) pot ser determinada amb quatre referències a la taula (veure figura 1.3.c):

$$\begin{aligned} Sum\,\mathrm{Re}\,c(r) = {} & SAT(x-1,y-1) + SAT(x+w-1,y+h-1) - \\ & - SAT(x-1,y+h-1) - SAT(x+w-1,y-1) \end{aligned} \qquad \textbf{(1.4)}$$

Aquest va ser el estudi presentat per P. Viola i M. Jones, amb una altre nomenclatura (s'ha usat la de R. Lienhart per mantenir la coherència en les equacions), però no es contemplava el fet de com calcular les característiques rotades que si fa afegir Lienhart.

Per el càlcul dels rectangles rotats 45º, es defineix una altra imatge auxiliar anomenada RSAT(x,y) (Rotated Summed Area Table). S'agafa la suma dels píxels del rectangle rotat 45º amb la cantonada de més a la dreta en la posició

(x,y) i s'ha d'escampar fins als límits de la imatge (veure figura 1.3.b). Formalment:

$$RSAT(x,y) = \sum_{x' \le x, x' \le x' - y - y'} I(x',y') \qquad \textbf{(1.5)}$$

Pot ser calculada amb dues passades per tots els píxels. El primer pas d'esquerra a dreta i de dalt a baix:

$$RSAT(x,y) = RSAT(x-1,y-1) + RSAT(x-1,y) + I(x,y) - RSAT(x-2,y-1) \qquad \textbf{(1.6)}$$

amb RSAT(-1,y) = RSAT(-2,y) = RSAT(x-1) = 0.
El segon pas d'esquerra a dreta i de dalt a baix, es calcula de la següent manera:

$$RSAT(x,y) = RSAT(x,y) + RSAT(x-1,y+1) - RSAT(x-2,y) \qquad \textbf{(1.7)}$$

La suma del rectangle rotat r = ((x,y),(w,h),45) pot ser determinada amb quatre referències a la taula (veure figura 1.3.d i figura 1.4):

$$\begin{aligned} Sum\,\mathrm{Re}\,c(r) = {} & RSAT(x+w,y+w) + RSAT(x-h,y+h) - \\ & - RSAT(x,y) - RSAT(x+w-h,y+w+h) \end{aligned} \qquad \textbf{(1.8)}$$

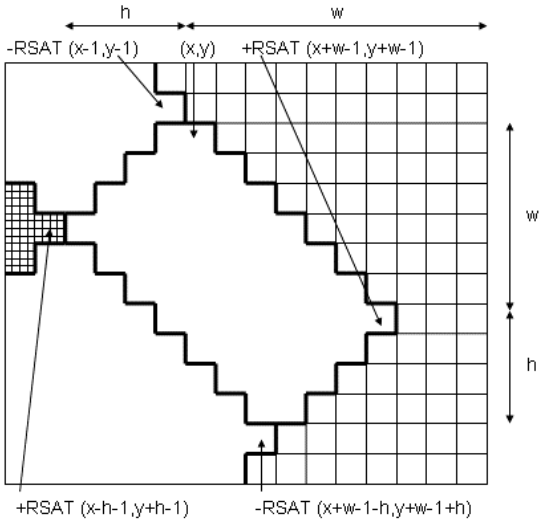


Fig. 1.4 Esquema per el càlcul d'àrees rotades

1.4. AdaBoost

El mètode AdaBoost, va ser presentat en 1995 per Y. Freund i R.E. Schapire. El seu nom es deu a que es un tipus de Boosting que s'ajusta adaptativament als errors de les característiques de classificació simples que genera el "Weak Learn" avaluada sobre la distribució de les mostres.

El AdaBoost pren com entrada un conjunt de m imatges d'entrenament, x_i, amb les seves corresponents etiquetes y_i (1 si es tracta d'una mostra positiva o -1 si es tracta d'una mostra positiva:

$$S = \left[(x_1, y_1),...,(x_m, y_m)\right] \quad \textbf{(1.9)}$$

Aquest mètode de Boosting accedeix a l'algoritme d'aprenentatge dèbil, "Weak Learn" (recordar que el seu objectiu es generar les característiques senzilles), mitjançant nombroses crides. En cadascuna de les quals s'avalua una característica simple en concret damunt de totes les mostres i per tant, se les classifica en funció de les etiquetes obtingudes per aquella característica simple. Així, les mostres queden distribuïdes en funció de la dificultat de classificació.

L'objectiu del "Weak Learn", com ja s'ha comentat, es trobar la hipòtesis que aconsegueixi minimitzar l'error entre la distribució de les mostres proporcionades. És a dir, si h_t representa una hipòtesi o característica simple pertanyent a la crida número t de l'algoritme d'aprenentatge, aleshores, l'error vindrà donat per:

$$\varepsilon_t = P_i\left[h_t(x_i) \neq y_i)\right] \quad \textbf{(1.10)}$$

on P_i es la probabilitat per a la mostra i de no pertànyer a la classe que ens afirma el classificador o hipòtesi dèbil.

L'algoritme per tant, en cada iteració, es quedarà amb la hipòtesi que millor classifiqui la distribució de mostres, és a dir, la que menor error presenti. En cada iteració el càlcul dels pesos de la distribució de mostres es torna a calcular. Aquest procés es repeteix T vegades fins a obtenir una combinació de T hipòtesis dèbils combinades en un únic classificador robust.

S'han proposat diferents algoritmes d'AdaBoost: Discrete AdaBoost (DAB), Real AdaBoost (RAB) i Gentle AdaBoost (GAB). La diferència que hi ha entre els tres sistemes, és la forma de actualitzar els pesos en cada iteració.

1.4.1. Discrete AdaBoost

Aquesta variant d'AdaBoost presentada, com ja s'ha comentat, per Y. Freund i R. Schapire es pot resumir de la següent manera:

- S'agafen un conjunt de imatges exemples $(x_1,y_1),...,(x_m,y_m)$ on y_i = -1,1 en funció si és un exemple negatiu o positiu.
- S'inicialitzen els pesos de les mostres que constitueixen els exemples amb $w_i=1/m$. Repetint per i=1,...,m. S'obté la distribució D.
- Iterar fins arribar al valor T (número de hipòtesis desitjades), on t és la iteració actual:

    - Crida al "Weak Learn" proporcionant la distribució de mostres.
    - Avaluar la hipòtesi dèbil h_t.
    - Calcular l'error per la hipòtesi h_t (equació 1.10).
    - Escollir el classificador h_t amb l'error més petit.
    - Actualitzar els pesos de la distribució D_t:

$$\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t} \qquad \textbf{(1.11)}$$

$$D_{t+1}(i,y) = \frac{D_t(i,y)}{Z_t}\beta^{\frac{1}{2}(1+h_t(x_i,y_i)-h_t(x_i,y))} \qquad \textbf{(1.12)}$$

    on Z_t es una constant de normalització escollida de manera que la suma de tots els pesos continuï valen la unitat.

- La hipòtesi final, o classificador robust és:

$$h_{final} = \arg\max \sum_{t=1}^{T} \alpha \cdot h_t(x,y) \qquad \textbf{(1.13)}$$

on $\alpha = \log(1/\beta_t)$.

L'avaluació de la hipòtesi final quan s'utilitzi, es compararà amb un llindar escollit en funció de les mostres del objecte proporcionades per decidir si una determinada subfinestra és objecte o no. A aquest llindar l'anomenarem *th*.

Com es pot observar, inicialment la distribució dels exemples és uniforme donat que tots els exemples presenten pesos iguals. Així, per obtenir la distribució de pesos que s'utilitzarà en la iteració t+1, es partirà de la que hi havia en la iteració anterior multiplicada pel valor β_t que farà que el valor del pes de la mostra x_i augmenti si h_t l'ha classificat incorrectament i disminueixi si l'ha

classificat correctament. La renormalització fa que els exemples senzills, els quals són correctament classificats per les primeres característiques simples, van quedant en un segon pla, per donar una major importància (pesos majors) a aquelles mostres que resulten difícils, i per tant incorrectament etiquetades pel classificador.

La hipòtesi final es el resultat d'un vot ponderat de la majoria de les característiques simples, és a dir, que per un exemple x, el classificador h_{final} obté com sortida l'etiqueta de la classe y, a la que pertany x en funció del seu criteri. El pes de cada característica simple h_t es defineix com α, per tant tindran majors pesos aquelles hipòtesis que hagin presentat menor error durant l'execució de l'algoritme. Finalment, h_{final} permetrà etiquetar el candidat com positiu o negatiu en funció de si l'avaluació proporciona un resultat pel damunt o per sota del llindar *th*.

La importància d'aquest algoritme, i de l'AdaBoost en general, resideix en el teorema que demostra que si l'error de cadascuna de les hipòtesis dèbils es tant sols lleugerament inferior a 0.5, aleshores l'error que tindrà la hipòtesi final h_{final} convergirà a zero exponencialment. Això suposa que les característiques simples han d'ésser simplement una mica millors que el que el resultat que s'obtindria al classificar de forma aleatòria, malgrat que com ja es veurà en el capítol 3, no serà suficient per obtenir un bon detector de cares en temps real.

1.4.2. Real AdaBoost

Aquesta variant de l'AdaBoost és molt semblada a l'anterior però amb una única diferència que és la manera d'actualitzar els pesos en cada iteració. L'actualització és la següent:

$$D_{t+1}(i,y) = D_t(i,y)\exp(y_i \cdot h_t(x_i, y_i)) \qquad \textbf{(1.14)}$$

tenint en compte en normalitzar tots el pesos quan s'assignin els seus nous valors.

1.4.3. Gentle AdaBoost

Aquesta variant d'AdaBoost va ser presentada en l'any 2002, per R. Lienhart i els seus col·laboradors, dintre del context de detecció d'objectes mitjançant un processat de les imatges a una alta velocitat. Com va reflectir en el seu article [6], van demostrar empíricament que aquesta varietat de Boosting podia superar les utilitzades per Freund - Schapire i Viola – Jones. Aquesta millora es fonamenta principalment en una càrrega computacional menor i una probabilitat de detecció major amb uns valors de velocitat similars.

Manté la mateixa filosofia que en els mètodes anteriors només variant la manera d'actualitzar els pesos:

$$D_{t+1}(i,y) = D_t(i,y)\exp(-y_i \cdot h_t(x_i, y_i)) \quad \textbf{(1.15)}$$

tenint en compte sempre la renormalització dels pesos.

1.5. Construcció de classificadors eficients amb AdaBoost

1.5.1. Cascada de classificadors

P. Viola i M. Jones van proposar un mètode per combinar classificadors cada vegada més complexos en una estructura en cascada o arbre jeràrquic, la qual incrementava dràsticament la velocitat de detecció, donat que aquesta estructura va centrant progressivament la seva atenció en les parts de la imatge més prometedores de contenir l'objecte a detectar.

Aquesta idea es basa en que en una imatge hi ha un número altíssim de subfinestres que no contindran l'objecte a detectar. D'aquesta manera es reserva el processat més complex i amb un cost computacional major només a les zones prometedores de contenir l'objecte. Els primers classificadors de la cascada són els encarregats de descartar la majoria de subfinestres que composen la imatge.

Una cascada de classificadors es un conjunt de classificadors senzills agrupats formant diferents etapes, dins d'aquestes, s'avaluaria cada subfinestra, si el resultat ens l'assenyala com a possible candidat a contenir l'objecte, aleshores passarà a ser avaluada per la següent etapa.

Les etapes que composen la cascada es construeixen amb classificadors entrenats mitjançant AdaBoost. L'anomenada cascada de classificació es coneguda com un arbre de decisió degenerat. Si els primers classificadors donen una resposta positiva a la subimatge d'entrada, comença el processat d'aquesta imatges amb la segona etapa, i així fins al final de l'arbre de decisió. Una resposta negativa condueix a un rebuig automàtic de la subimatge com es pot veure en la figura 1.5.

L'estructura de la cascada reflexa el fet de que dins d'una única imatge, la majoria de les subimatges seran classificades com negatives. D'aquesta manera, la cascada tendeix a rebutjar a la majoria de candidats negatius d'una imatge en les etapes inicials. Mentre que el fet de que una subfinestra passi satisfactòriament totes les etapes es considera un fet poc probable.

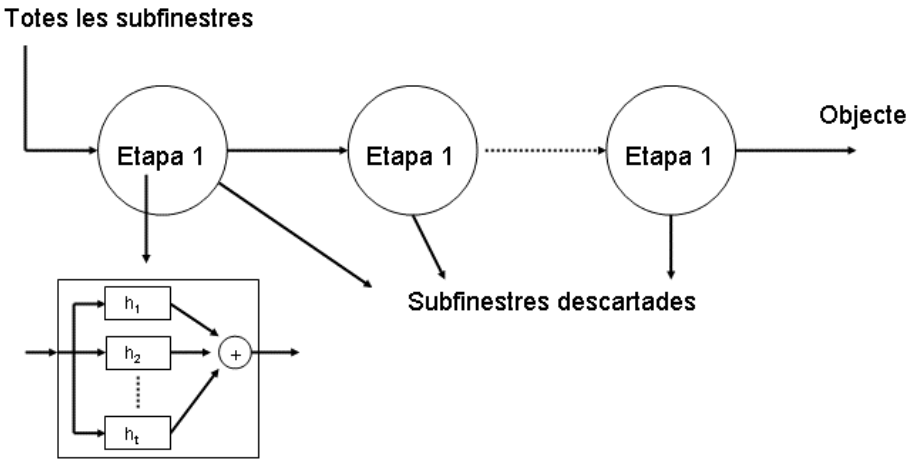


Fig. 1.5 Cascada de classificadors

Per construir las cascada de classificació d'aquesta manera, els classificadors de les etapes finals, se entrenen utilitzant les mostres negatives que les etapes anteriors han classificat com positives quan realment no ho són, aquestes són més difícils de classificar que les típiques. Amb lo que cada etapa posterior realitza una tasca més complexa que l'etapa anterior més propera.

A més, s'ha de tenir en compte que la probabilitat de trobar un objecte en la imatge, és molt baixa fent que l'opció més fàcil sempre sigui descartar la imatge com no candidata a contenir l'objecte a detectar més que acceptar-la com a objecte.

Aquesta estructura dona una major robustes al sistema de detecció ja que es realimenta durant l'entrenament amb els exemples negatius que més costen de classificar com a negatius.

El procés de disseny o entrenament d'aquesta cascada de classificadors, es caracteritza per unes determinades taxes de detecció i de falsa alarma que es volen aconseguir. Es parteix dels valors de la probabilitat de detecció i falsa alarma que volem que tingui el classificador. En sistemes de detecció de cares anteriors, basats en altres mètodes, s'havien aconseguit bones taxes de detecció (entre el 80% i el 90%) i de falsa alarma (del ordre de 10^{-5}). El número de etapes que formen la cascada de detecció i el número de característiques simples que composen cada etapa deuen ser suficients per aconseguir unes taxes similars (tenint en compte que amb aquest sistema aconseguim minimitzar el cost computacional i per tant, el temps de processat).

Donada una cascada de classificadors, la taxa de falsa alarma de tota la cascada vindrà donada per:

$$F = \prod_{i=1}^{K} f_i \quad \textbf{(1.16)}$$

on K és el número total d'etapes i f_i la taxa de falsa alarma de cadascuna de les etapes per separat.

La taxa de detecció serà:

$$F = \prod_{i=1}^{K} d_i \quad \textbf{(1.17)}$$

on K també és el número total d'etapes i d_i la taxa de detecció de cadascuna de les etapes per separat.

Així, al concretar els valors que volem aconseguir per les probabilitats de detecció i falsa alarma, podrem saber el que tindrà que complir cada etapa per separat. Per exemple, una taxa de detecció de 0.9 es pot aconseguir amb 10 etapes que presentin una taxa de detecció de 0.99 ($0.99^{10} \approx 0.9$). Aquest fet sembla contradictori ja que sembla que amb una sola etapa podem aconseguir millors taxes de detecció, però s'ha de tenir en compte que amb una etapa únicament obtindríem taxes de falsa alarma del 20-30% i és clar, amb un número molt elevat de hipòtesis dèbils entrant en conflicte amb el fet que es vol realitzar un classificador per un sistema en temps real. Això si, amb una taxa de falsa alarma del 30% i 10 etapes tindríem una taxa global de $6 \cdot 10^{-6}$ (0.3^{10}) que realment ja són uns resultats a tenir en compte. Per aquest motiu es necessària la concatenació de les successives etapes.

El procés que permet el entrenament de les etapes en la cascada requereix una gran meticulositat ja que se ha observat que es molt sensible a certs paràmetres que poden fer que s'obtinguin classificadors molt diferents en quant a les seves taxes de detecció i falsa alarma. La majoria de classificadors en cascada amb un número elevat de característiques simples, aconsegueixen millors resultats però amb un cost computacional major. En principi els paràmetres més crítics en la construcció d'una cascada de classificadors són:

- El número de etapes de la cascada.
- El número de hipòtesis dèbils que formen cada etapa per separat.
- El llindar de cada etapa (th).

P. Viola i M. Jones van utilitzar un marc realment simple per a construir un classificador altament eficient. L'esquema fet servir en seu treball va ésser el següent:

- S'ha de seleccionar els valors per f_i, d_i, F i D.
- Es comença amb dos conjunts de mostres, un amb mostres positives i un amb mostres negatives, P i N respectivament.

- S'inicialitza F_0=1, D_0, i=0.
- Mentre F_i>F:
    - Incremetar i
    - $n_{hipòtesisdèbils}$ =0, F_i= F_{i-1}
        - $n_{hipòtesisdèbils}$= $n_{hipòtesisdèbils-1}$
        - Fem servir P i N per entrenar un classificador amb $n_{hipòtesisdèbils}$ mitjançant AdaBoost
        - Avaluem la cascada obtinguda mitjançant el conjunt de mostres de exemple per determinar F_i i D_i
        - Es disminueix el llindar de l'etapa i fins que el classificador actual tingui una taxa de detecció d'almenys d_ixD_{i-1}
    - Si F_i>F aleshores avaluar las cascada de classificació actual sobre el conjunt de imatges que no contenien l'objecte i col·locant també subfinestres classificades erròniament, és a dir que han suposat falses alarmes.

1.5.2. Arbre de detecció.

En el subapartat anterior, veiem com en una estructura en cascada, per cada etapa, un cert percentatge de candidats eren descartats i la resta passaven a la següent etapa. P. Viola i M. Jones van tenir molt d'èxit amb aquesta estructura aplicada a la detecció de cares frontals. Després d'aquests resultats, R. Lienhart i d'altres (veure [5]) van voler arribar una mica més lluny amb una estructura que permetés construir detectors orientats a objectes amb una major variabilitat. Varen dissenyar una nova estructura que es pot interpretar com un arbre on les branques estan formades per cascades de classificadors especialitzats com varies etapes de detecció, cadascuna formada per varis classificadors o característiques simples, sempre que això suposes una millora en la detecció sense augmentar el cost computacional, veure figura 1.6.

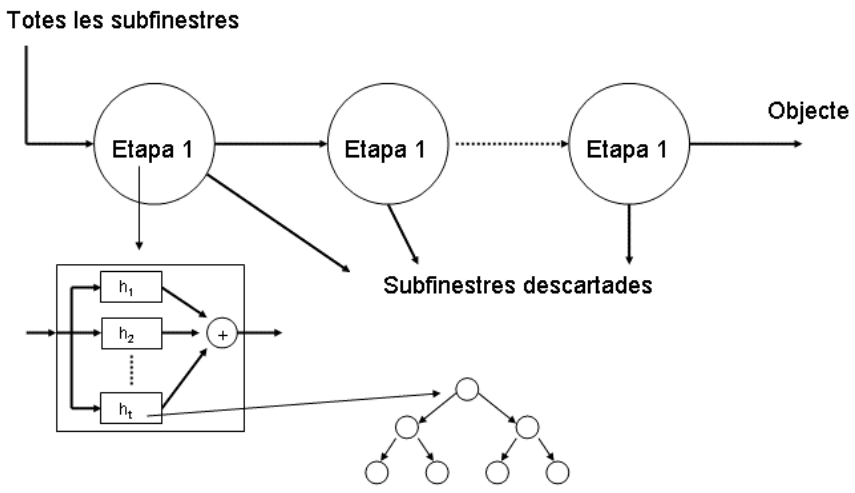


Fig. 1.6 Arbre de detecció

L'entrenament d'un arbre d'aquestes característiques partiria del node arrel. Aquest node es diferencia de la resta de nodes en que no depèn de cap altre node pare. El conjunt de mostres positives d'entrenament pel node arrel serà el conjunt de mostres positives al complert, és a dir, el conjunt de totes les mostres positives de que es disposa per tot l'entrenament.

L'algoritme proposat per R. Lienhart és un procediment recursiu. En cada node, totes les mostres positives i negatives trameses pel node pare, s'utilitzen per entrenar un classificador mitjançant AdaBoost. La complexitat computacional està linealment relacionada amb el número de classificadors dèbils que composen cada branca , que es una cascada de classificadors tal i com s'ha vist en el subapartat anterior. Després d'entrenar el node pare, l'arbre es comença a dividir en branques, anomenades splits. Per l'entrenament de cadascuna de les branques, es farà servir el total de exemples negatius i la part de exemples positius resultant de dividir el total de exemples per les rames que s'utilitzaran. Si el número total de característiques de tots els classificadors que composen l'arbre es menor que el número de característiques utilitzades per construir un classificador del tipus explicat en el subapartat anterior, aleshores aquesta estructura serà més eficient computacionalment. L'algoritme recursiu de construcció de les branques es repetirà successivament fins que s'aconsegueixi la profunditat sol·licitada pels objectius de detecció especificats.

Una vegada construït l'arbre, el procés de classificació d'una subfinestra serà el següent: el candidat començarà a ésser avaluat pel node pare, si aquest el descarta, la subimatge serà etiquetada com negativa i el procés finalitzaria. Si per el contrari el node arrel l'accepta, aquesta passarà a una de les branques en que s'haurà dividit el node pare. Si la branca la classifica com a negativa passarà a ser avaluada per una altre branca, i així fins trobar un camí d'acceptació que permetrà a la subfinestra arribar al final de l'arbre i passar a la següent etapa. Si les supera totes serà classificada com positiva. Si per el contrari, en algun punt de l'arbre, ja estan explotats tots els camins possibles i la subfinestra ha sigut descartada per tots, serà etiquetada com negativa i s'acabarà el procés.

R. Lienhart i el seu equip van demostrar que amb aquesta estructura s'aconsegueix una taxa de detecció i de falsa alarma de:

$$F \leq K \cdot f_i^K \qquad \textbf{(1.18)}$$

$$D \leq K \cdot D_i^K \qquad \textbf{(1.19)}$$

on K continua sent el número d'etapes. Aquest increment de la taxa de falsa alarma respecte amb una cascada de classificadors, es pot compensar entrenant un número superior d'etapes, concretament:

$$\Delta K = \frac{\log\left(\frac{1}{K}\right)}{\log(f_i)} \qquad \textbf{(1.20)}$$

Es lògic pensar que així augmentarem el cost computacional però a la pràctica no es així donat que les etapes extres s'avaluaran en rares ocasions així que la seva contribució al cost es podrà considerar menyspreable.

Malgrat que s'ha de tenir en compte que per considerar un arbre de detecció com a millor que una cascada de classificadors, el número de característiques emprades ha d'ésser igual o menor que les usades en una cascada de classificadors.

CAPÍTOL 2. APLICACIÓ DEL BOOSTING

2.1. Introducció

Després de veure la part teòrica, en aquest capítol s'explicarà el procés que s'ha seguit per construir un estructura de arbre de classificació però només amb una branca. Per aquesta finalitat, s'ha utilitzat part d'un paquet d'aplicacions integrat en la llibreria de Intel Open CV Beta 5 ®, escrita en C++, i d'altres aplicacions realitzades en aquest estudi per poder realitzar tot el procés d'una manera eficient.

El procés per aconseguir realitzar el classificador i comprovar el seu funcionament consta de 3 blocs ben definits (veure figura 2.1.).

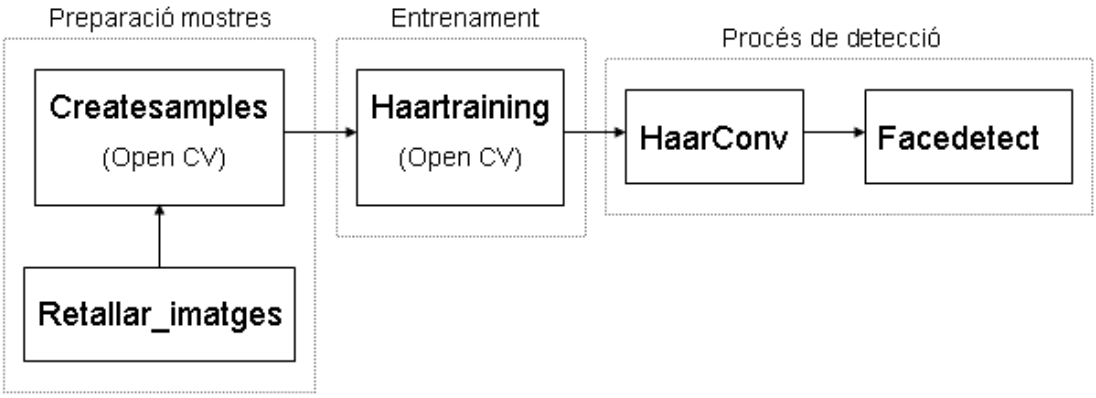


Fig. 2.1 Diagrama de blocs

La primera part consisteix en l'elecció de les mostres, tant positives com negatives. L'adequació de les mostres positives a la grandària d'entrenament (24x24) mitjançant l'aplicació *retallar_imatges* feta amb MatLab per aquest ús. En la integració amb l'aplicació *createsamples* per ser utilitzades en l'entrenament del classificador ja que l'aplicació d'entrenament necessita varis fitxers amb el llistat de la ubicació i nom de les mostres, tant positives com negatives.

En a segona part s'explicarà el funcionament de l'aplicació *haartraining* que és la que se encarrega de realitzar l'entrenament del classificador.

Per finalitzar, s'explicarà el procediment per realitzar les aplicacions necessàries per realitzar el procés de detecció: *haarconv* i *facedetect* i a més poder comprovar l'eficiència de classificador realitzat.

2.2. Procés obtenció i preparació de les imatges per l'entrenament

Aquesta primera fase del procés és la part que més canvis ha anat patint des de l'inici del treball. Sobretot per l'experiència obtinguda amb els resultats dels classificadors que s'anaven aconseguint.

Com ja va quedar assenyalat en el capítol 1, l'avantatges del mètode de Boosting consisteix en trobar les característiques que millor classifiquin l'objecte a detectar, així com combinar-les de la manera més eficient possible. Per aquest motiu aquesta etapa inicial serà la clau de l'èxit per trobar les millors característiques possibles ja que l'algoritme d'entrenament té una gran sensibilitat i una incorrecte selecció de les mostres a utilitzar pot portar al fracàs a l'hora d'obtenir un classificador amb unes bones taxes de detecció i falsa alarma.

2.2.1. Exemples positius

El primer pas que es va realitzar va ser la cerca aleatòria en la world wide web del major número de imatges possibles que continguessin cares frontals, la majoria imatges RGB i compressió jpeg. Com a mínim es volia tenir al final del procés un número molt elevat de imatges per obtenir després unes 5000 mostres positives adequades per l'entrenament del classificador, el mateix número de mostres fet servir per P. Viola i M. Jones.

Per extreure les mostres positives de les imatges d'exemple, es va fer una aplicació amb MatLab, anomenada retallar_imatges, on es trobaven les coordenades dels ulls i després es feia el processat corresponent per obtenir les mostres a la grandària d'entrenament. Es va fer servir MatLab ja que té una eina molt potent per trobar coordenades dins d'una imatge. Aquesta eina s'anomena "control point selection tool", és molt potent però s'ha de tenir cura amb el seu ús donat que si no es programa correctament, veure plana web MatLab, no proporcionarà les coordenades correctament.

L'aplicació es basa en trobar la distància entre els ulls i realitzar un redimensionat de la imatge a la grandària d'entrenament.

L'aplicació realitzada és molt senzilla d'utilitzar, es treballa amb dues imatges simultàniament i s'ha de marcar primer un punt en una imatge, després un altre punt en la segona imatge, torna a marcar a la primera i després finalitzar amb el darrer punt a la segona imatge, és a dir, s'ha de alternar de imatge cada vegada que es marca un punt, veure figura 2.2.

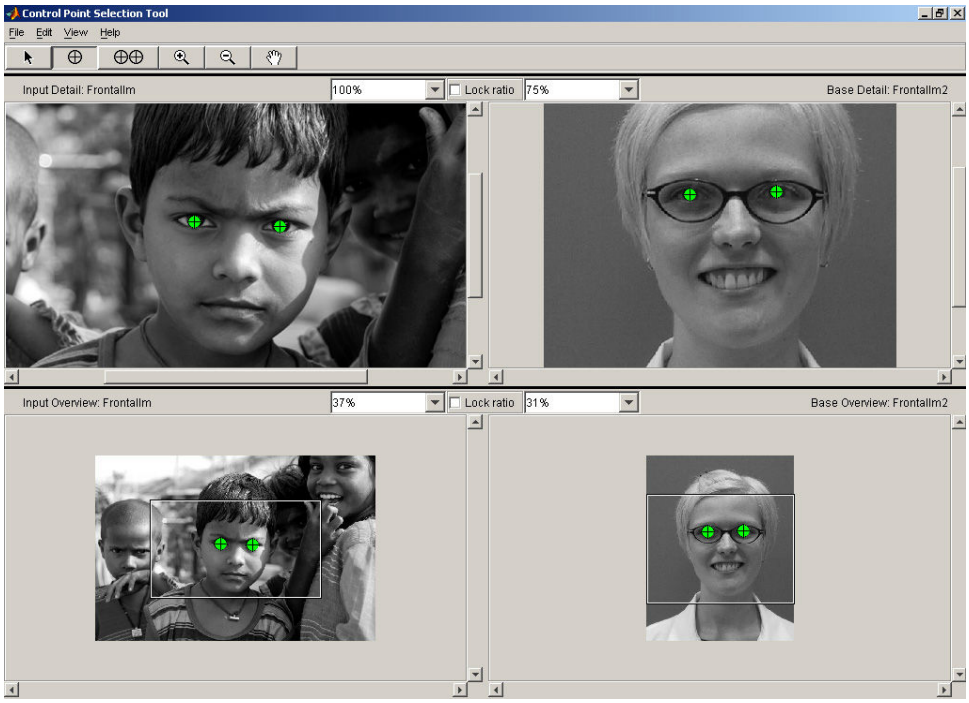


Fig. 2.2 Imatge de l'eina controlpointselectiontool de MatLab amb els punts ja marcats.

Un cop ja es tenien les distàncies entre els ulls, es normalitzen en funció de la distància entre els ulls que volíem tenir en la mostra positiva de 24x24 píxels, després de moltes proves, es va veure que la distància òptima tenia que ser 1/3 de l'amplada de la imatge d'entrenament, en aquest cas 8 píxels, veure figura 2.2, dreta. S'obtenia un factor que s'utilitza per redimensionar la imatge original a la grandària adequada per obtenir una mostra positiva on la cara quedés perfectament enquadrada en una finestra de 24x24 píxels. S'utilitza per redimensionar la imatge la transformació bicúbica. Un cop redimensionada, només quedava retallar-la en funció de les distàncies adequades per tenir la cara centrada. Després de successives proves es va arribar a la conclusió que referència amb l'ull esquerra, les distàncies tenien que mesurar: a) 1/3 de la grandària d'entrenament, b) 2/3, c) 1/3 i d) 2/3. Veure figura 2.3, esquerra.

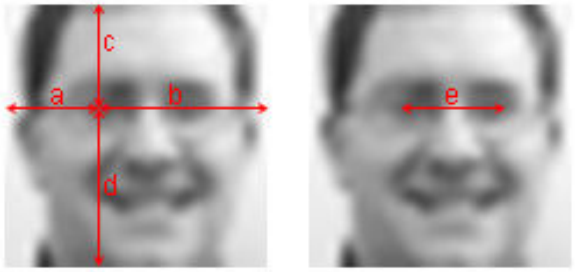


Fig. 2.3 A l'esquerra tenim les distàncies de referència utilitzades per retallar les mostres i a la dreta s'assenyala quina és la distància per normalitzar la imatge.

Es va tenir cura en tot moment que el redimensionat fos sempre a menor grandària, evitant així treballar amb imatges més petites de 24x24 píxels (mostra d'entrenament). Qualsevol imatge original que no complís aquesta premissa o que no fos possible encabir en 24x24 píxels (per exemple, perquè es trobés molt propera als límits de la imatge) passaria a un directori de rebuig. Les imatge originals processades passarien a un directori com ja tractades i les mostres generades passarien al directori de mostres positives.

Les mostres obtingudes estan en escala de grisos i emmagatzemades en format mapa de bits (.bmp). Recordar que qualsevol imatge original pot ser tant RGB com escala de grisos i en format jpg, bmp i tiff.

Una vegada obtinguda la col·lecció complerta de les imatges positives per l'entrenament, es tenia que crea un fitxer de text (l'extensió del mateix té que ser .idx), necessari per l'aplicació d'entrenament, on s'especificava el path i el nom complert de cadascuna de les imatges de la següent manera:

Exemple fitxer: nom_fitxer_ex_positius.idx

c: \positives\1.bmp
c: \positives\10.bmp
c: \positives\100.bmp
c: \positives\1003.bmp
c: \positives\1004.bmp

Una manera molt senzilla de realitzar aquest fitxer es, des de l'intèrpret de comandes de windows, ubicar-se en el directori on es troben les imatges (cd path_ubicació_imatges) i executar la comanda: dir /b /s > c:\path_de_destí del_fitxer\nom_fitxer_ex_positius.idx
S'ha de tenir en compte que si el path de destí és el mateix on es troben les imatges, el path i el nom del fitxer apareixeran en el llistat generat.

Amb aquest pas no n'hi ha prou per tenir el fitxer preparat per la utilització amb l'aplicació createsamples.

En cada línia de text, es té que afegir darrere del nom el número de cares que apareixen en la imatge i després les coordenades del rectangle on es troba cadascuna de les cares (x, y, amplada, altura).

Com ja es veurà més endavant per evitar degradacions de les mostres, es van proporcionar les imatges amb la grandària exacte per l'entrenament i única i exclusivament amb un cara per imatge. Aleshores, només tindrem que afegir a cada línia del fitxer: 1 0 0 24 24

Exemple fitxer modificat: nom_fitxer_ex_positius.idx

c: \positives\1.bmp 1 0 0 24 24
c: \positives\10.bmp 1 0 0 24 24
c: \positives\100.bmp 1 0 0 24 24
c: \positives\1003.bmp 1 0 0 24 24

c: \positives\1004.bmp 1 0 0 24 24

Aquest darrer pas es va fer amb una petita aplicació escrita amb llenguatge C++ anomenada arxiu.

En aquest punt ja tenim els exemples positius i el fitxer preparats per ser proporcionats a l'aplicació d'entrenament.

2.2.2. Exemples negatius

Com es veurà en el següent subapartat, les imatges de fons, és a dir, imatges sense cares, no tenien que tenir un tractament especial. No calia retallar-les a la grandària adequada per l'entrenament, serà la mateixa aplicació d'entrenament qui s'encarregarà d'obtenir les mostres amb grandària adequada. Això si, sempre tenint cura que no hi hagués cap cara en les imatges ja que aquest fet evidentment ens faria entrenar un classificador erroni.

El fet més important respecte a l'obtenció dels exemples negatius és la variabilitat de les imatges, és a dir, obtenir la màxima informació possible de les mateixes. Per tant, la cerca de les imatges venia condiciona per aquest fet, s'havia d'aconseguir una col·lecció de imatges prou gran però sempre amb imatges molt diferents entre elles, quan més millor. Aquest punt quedarà més clar en el apartat 2.3.

Una vegada obtinguda la col·lecció complerta de les imatges negatives per l'entrenament, es tenia que crea un fitxer de text (l'extensió del mateix té que ser .idx), necessari per l'aplicació d'entrenament, on s'especificava el path i el nom complert de cadascuna de les imatges de la següent manera:

Exemple fitxer: nom_fitxer_ex_negatius.idx

c:\negatives\00_airplane.jpg
c:\negatives\05_05_18_piorunochron.jpg
c:\negatives\1111_little_italy.jpg
c:\negatives\Amicon.jpg
c:\negatives\amish-cigars.jpg

Una manera molt senzilla de realitzar aquest fitxer es, des de l'intèrpret de comandes de windows, situar-se en el directori on es troben les imatges (cd path_ubicació_imatges) i executar la comanda: dir /b /s > c:\path_de_destí del_fitxer\nom_fitxer_ex_negatius.idx

S'ha de tenir en compte que si el path de destí és el mateix on es troben les imatges, el path i el nom del fitxer apareixeran en el llistat generat.

En aquest punt ja tenim els exemples negatius i el fitxer preparats per ser proporcionats a l'aplicació d'entrenament.

2.2.3. Opcions de createsamples

Aquesta aplicació creada per Intel, és l'eina que ens permet generar el fitxer de mostres positives per l'entrenament del classificador. S'encarrega de generar un fitxer del tipus vector (.vec) que utilitzarà posteriorment l'entrenador.

Es pot fer de dues maneres, la primera es partint d'un conjunt de imatges i la segona es partint d'una única imatge (ex: logo). Amb aquesta segona opció l'aplicació generarà una col·lecció des d'aquesta única imatge mitjançant rotacions, canvis de colors, distorsions.

Aquest no és el nostre cas per lo tant només es farà referència a la primera opció sense esmentar les crides del programa relacionades amb aquesta darrera manera d'operar.

La crida que en tot el treball s'ha realitzat ha estat:

createsamples -info nom_fitxer_ex_positius.idx -vec nom_fitxer_ex_positius.vec -num número_de_mostres

-info: precedeix al nom del fitxer on està el llistat de les imatges de totes les imatges positives.
-vec: darrera escriurem el nom del fitxer .vec que generarà l'aplicació a partir de les mostres proporcionades.
-num: darrera posarem el número de mostres que ha de generar el programa, evidentment té que ser menor o igual que el número de mostres proporcionades.
-w: hem de posar l'amplada que volen per les mostres. Per defecte és 24 píxels.
-h: hem de posar l'alçada que volen per les mostres. Per defecte és 24 píxels.
-show: si posem aquesta opció s'aniran veient per pantalla totes les mostres. Es pot sortir en qualsevol moment d'aquesta visualització dins del programa mitjançant escape.

S'havia comentat anteriorment que les mostres positives les proporcionàvem directament amb la grandària adequat per evitar possibles degradacions de la qualitat de les mateixes. Aquest fet es degut que dins de l'execució de l'aplicació *createsamples* es fa una crida a una funció anomenada cvCreateTrainingSamplesFromInfo que s'encarrega de redimensionar les mostres a la grandària sol·licitada, i és clar, no volíem mai redimensionar una mostra a major grandària.

2.3. Entrenament del classificador

2.3.1. Procés d'entrenament

Un cop ja hem preparat les imatges i els fitxers corresponents, ja es pot iniciar l'entrenament d'un classificador. Per aquesta tasca també treballarem amb una aplicació realitzada també per Intel anomenada *haartraining*.

És una aplicació que creada amb objecte de investigar sobre els algoritmes de boosting aplicats a la detecció d'objectes en sistemes en temps real, concretament en cares humanes. Tot el codi font es pot descarregar des de la plana web de Intel i de codi obert. Aquest codi inclou l'aplicació principal junt amb altre fitxers de llibreries de funcions tant de Boosting com d'altres tasques pel tractament de imatges. Els fitxers de codi utilitzats per l'entrenament són:

- haartraining.cpp (fitxer que conté l'aplicació principal)
- cvhaartraining.cpp
- cvhaarclassifier.cpp
- cvboost.cpp
- cvhaar.cpp

A més és necessari usar les llibreries cxcore.lib, cv.lib per poder arribar a tenir èxit en l'execució i compilació de l'aplicació.

El programa d'entrenament, *haartraining*, implementa tot un procés de construcció dels classificadors d'objectes que es considerablement complex tant d'entendre com d'executar. Aquesta aplicació presenta un cost computacional molt elevat, tant, que depenent de les grandàries dels objectes, característiques emprades, número d'etapes, pot arribar a durar més d'una setmana.

Haartraining.cpp és la aplicació de interfície entre l'aplicació real de Boosting i l'usuari. S'encarrega única i exclusivament de recollir tots els paràmetres que ha introduït l'usuari per configurar l'execució del programa que realment s'executa amb la funció cvCreateTreeCascadeClassifier. Aquesta funció és la que s'encarrega de generar el classificador. La descripció de les tasques que realitza la descriurem d'una manera general sense entrar en detall en totes les funcions que intervenen en l'aplicació:

1. Reserva de memòria: en primer lloc, es fa la reserva de memòria necessària per allotjar a tot el classificador complert, és a dir, amb el número d'etapes sol·licitades. S'inicialitza l'estructura anomenada cvCascadeHaarClassifier que contindrà el classificador, a més s'assignen les funcions encarregades d'avaluar tota la cascada amb les imatges i d'alliberar la memòria reservada al final de tot el procés. La funció encarregada d'aquesta tasca es icvCreateCascadeHaarClassifier.

2. Imatges de fons: s'obre el fitxer on s'especificava el path i el nom de les imatges negatives, es conten i s'emmagatzemen en un vector. Aquesta tasca correspon a les funcions icvInitBackgrounReaders (que inizialitza

les estructures per les mostres) i icvCreateBackGroundData (que s'encarrega de la tasca d'emmagatzemar les imatges).

3. Matriu d'entrenament: totes les mostres, tant positives com negatives que es faran servir per construir el classificador, s'emmagatzemen en una gran estructura de matrius i dades numèriques. La grandària de les matrius vindrà donat pel número de mostres que s'hauran proporcionat. També inclou els pesos. Aquesta estructura de dades és CvHaarTrainingData i és la que es farà servir per tot l'entrenament per extreure tota la informació per l'entrenament.

4. Característiques simples: es realitza la construcció de totes les característiques simples possibles dins de la grandària de la mostra. La funció encarregada de fer aquesta tasca és icvCreateIntHaarFeatures.
   En aquest moment ja s'està preparat per començar el procés d'entrenament, a partir d'ara entrem en un bucle on en cada iteració es crearà una etapa (o es carregarà si l'etapa ja existeix).

5. Comprovació si ja existeix una etapa prèvia: si existeix part d'un classificador realitzat, l'aplicació pot continuar l'entrenament des de la darrere etapa realitzada. Les carrega mitjançant la funció icvLoadCARTStageHaarClassifier, i continua per l'etapa immediatament següent. Si no existeix crea el directori on s'emmagatzemarà l'etapa en curs.

6. Obtenció mostres positives: s'obre el fitxer que conté les mostres positives en format vector, comprova el seu número i grandària . En aquest moment va avaluant una amb una cada mostra per tenir en compte només les que donen resultat diferent de zero.. La funció principal que s'encarrega d'aquesta tasca es icvGetHarrTrainingDataFromVec.

7. Cerca de candidats negatius: els candidats negatius s'extreuen des del fitxer proporcionat amb les imatges de fons. Tindran la mateixa grandària de les mostres positives. Per fer la cercar va carregant les imatges del fitxer una a una. A partir del punt inicial (x=0,y=0) de la imatge s'agafa una subfinestra de la grandària de la mostra i s'avalua amb la cascada de classificació si es que hi ha alguna etapa ja creada. Si el resultat es diferent de zero, aquesta subfinestra es tractarà com una mostra negativa. Després continua amb la següent subfinestra que en aquest cas serà (x=0+ample mostra+1,y=0) i es farà la mateixa acció d'avaluar i computar si s'escau. Aquest procés es fa per tot x i per tot i fins escombrar tota la imatge i passarà a una altre imatge. La cerca finalitzarà quan obtinguem tots el candidats necessaris per l'entrenament de l'etapa en curs. Si es tingués que tornar a començar per la primera imatge el punt inicial serà (x=1,y=1) evitant així repetir imatges ja utilitzades. S'ha de tenir en compte que el algoritme emmagatzema en quina posició es va quedar en l'última cerca per així continuar pel mateix punt en la propera sol·licitud d'exemples negatius.

Es va fer un seguiment al procés exacte i es va veure a mesura que avançava el procés d'entrenament l'hi era més difícil a l'aplicació obtenir mostres noves per realitzar l'entrenament. Això va fer deduir que el fet de tenir una col·lecció de imatges molt gran no volia dir que fossin les suficients pel procés d'entrenament si no que era millor tenir una col·lecció de imatges potser més petita però amb una major variabilitat entre elles per així proporcionar a l'algoritme d'entrenament major quantitat d'informació i no única i exclusivament gran quantitat de imatges.

8. Assignació de pesos: com ja es va dir en la descripció del mètode de boosting, es necessari anar donat més importància a unes mostres o a d'altres en funció de la seva dificultat per ésser classificades. Inicialment els pesos valen:

$$pes = \frac{1}{n^{o}\_mostres\_negatives + n^{o}\_mostres\_positives} \qquad \textbf{(2.1)}$$

Per qualsevol etapa posterior, es tindran que tornar a assignar, la funció que s'encarrega d'aquesta tasca és icvPreculate que utilitza les etapes ja construïdes per avaluar totes les mostres i ordenar-les en funció de la dificultat per la seva correcta classificació.

9. Gravar etapa creada en un fitxer de text. S'emmagatzema en el directori corresponent a l'etapa, amb el nom AdaBoostCARTHaarClassifier.txt.
La manera com queda emmagatzemada l'etapa es la següent:

```
2
1
2
8 8 2 4 0 -1
8 10 2 2 0 2
haar_y2
3.763943e-003 0 -1
-8.524691e-001 7.104851e-001
1
2
9 6 8 4 0 -1
11 6 4 4 0 2
haar_x4
6.597492e-003 0 -1
-8.572406e-001 5.786675e-001
```

El primer número ens assenyala la quantitat de característiques que s'utilitzen en aquesta etapa. Segueix sempre amb un 1 i a continuació el número de rectangles fets servir en la característica concreta. Després s'indica x_{origen}, y_{origen}, amplada, altura, 0 i pes per cada rectangle. La següent línia ens indica el nom de la característica. Després tenim el

llindar conjunt de la característica i dues xifres que ens indiquen el camí a seguir en funció de si es supera el llindar o no (recordem que s'està construint un arbre de classificació però amb una única branca) , el número 0 ens assenyala que una branca no està activa quedant només activa la branca -1. Al final tenim els llindars individuals per cadascun dels rectangles.

10. Fi del procés: si ja s'ha creat la darrere etapa, l'aplicació finalitza l'execució de l'entrenament alliberant la memòria que s'hagi estat utilitzat en el procés.

2.3.2. Factor de Trimming

El trimming es un mètode fet servir en l'aplicació d'entrenament per eliminar les mostres que menor informació aporten a l'entrenament. Recordem que l'algoritme d'entrenament s'encarregava de tornar a assignar els pesos en funció de com es classificava una mostra fent els pesos de les imatges que millor es classificaven més petits. Fent servir l'opció de trimming s'eliminen aquestes imatges que gairebé no proporcionen informació nova per la creació de posterior etapes. El factor d'eliminació va donat per l'opció en la crida al programa, que posteriorment veurem, anomenada weightfraction. El valor possible d'aquesta opció a d'estar en 0 i 1, quan més proper a zero, més mostres es descartaran en el procés d'entrenament. El valor que ve donat per defecte, 0.95, en el nostre cas va funcionar correctament i és clar, redueix lleugerament el temps d'entrenament ja que descarta mostres que són redundats per la seva facilitat de classificació. La funció que s'encarrega d'aquesta opció es cvTrimWeights.

2.3.3. Normalització de les imatges

Des de la presentació del treball de P. Viola i Jones "Robust Real-time Object Detection" (veure [3]) ja es va considerar la normalització de la variança a totes subfinestres de entrenament per minimitzar els efectes de diferents condicions d'il·luminació. La variança de una imatge pot ser ràpidament calculada mitjançat un parell de imatges integrals.

$$\sigma^2 = m^2 - \frac{1}{N}\sum x^2 \qquad \textbf{(2.2)}$$

on σ és la desviació estàndard, m la mitja i x és el valor del píxel.
En l'aplicació d'entrenament haartraining també s'utilitza la normalització. Aquest fet es realitza primer amb la funció cvIntegral de la llibreria Open CV on proporcionat una imatge, obtenim la imatge integral i la imatge integral dels píxels al quadrat. El procediment posterior dintre de la aplicació consisteix:

$$normfactor = \sqrt{area \cdot valsqsum - valsum \cdot valsum} \quad \textbf{(2.3)}$$

on àrea es l'àrea total, valsum és la imatge integral i valsqsum és la imatge integral dels píxels al quadrat.

Aquesta tasca de normalització prèvia, la realitza la funció icvGetAuxImages.

2.3.4. Opcions de haartraining

-data: s'utilitza per especificar el directori on s'emmagatzemarà el classificador. S'ha de tenir cura en finalitzar el nom del directori a usar amb el caràcter / , si no el gravarà en el directori des de on s'executi l'aplicació.
-vec: aquí especifiquem el nom del fitxer amb extensió .vec amb les mostres positives.
-bg: especificarem el fitxer que conté el path i el nom de les imatges d'on s'obtindran les mostres negatives.
-npos: número de mostres positives que s'utilitzaran per l'entrenament. Per defecte 2000.
-nneg: número de mostres negatives que s'utilitzaran per l'entrenament. Per defecte 2000.
-nstages: número de etapes que volem que tingui el nostre classificador. 14 per defecte.
-nsplits: número de branques o nodes. Per defecte 1
-mem: memòria utilitzada que es farà ús en el programa. Per defecte 200MB.
-sym: aquesta opció s'ha de activar si l'objecte a detectar presenta simetria vertical. És l'opció per defecte.
-nonsym: l'opció per anular la simetria vertical
-minhitrate: mínima taxa de detecció per cadascuna de les etapes. Per defecte 0.995.
-maxfalsealarm: màxima taxa de falsa alarma per cadascuna de les etapes. per defecte 0.5.
-weightfraction: factor de trimming. Per defecte 0.95.
-mode: aquí podem escollir els grups de característiques a usar. Hi ha 3 opcions: BASIC (per defecte), CORE i ALL.
L'opció BASIC inclou les característiques: haar_x2, haar_x3, haar_x4, haar_y2, haar_y3, haar_y4 i haar_x2_y2 (combinació de haar_x2 i haar_y2). L'opció CORE afegeix la característica anomenada point i per finalitzar l'opció ALL, són totes les anteriors més elles mateixes rotades 45º a dretes.
- w: amplada de la mostra d'entrenament. Per defecte 24.
- h: altura de la mostra d'entrenament. Per defecte 24
- bt: amb aquesta opció es selecciona el tipus de Boosting que volem fer servir per la construcció de classificador <DAB | RAB | LB | GAB (per defecte)>
    - DAB: Discrete AdaBoost
    - RAB: Real AdaBoost
    - LB: Logit Boost
    - GAB: Gentle AdaBoost

La crida que s'ha usat per realitzar l'arbre de detecció es a següent:

haartraining -data classificador/ -vec cares.vec -bg bg.idx -npos 3325 -nneg 4500 -maxfalsealarm 0.30 -nonsym -mem 400 –nstages 11

La resta d'opcions que s'activen per defecte ja eren correctes.

2.4. Detecció mitjançant el classificador

2.4.1. Introducció

Una vegada finalitzat el procés d'entrenament i amb un classificador ja realitzat es va procedir a crear l'aplicació per realitzar la detecció i així comprovar l'eficiència del classificador, en definitiva per poder ser testejat.

El primer problema que va aparèixer és que per poder treballar d'una manera més ràpida i eficient amb els prototips de càrrega de la cascada de classificació proporcionats per Intel es tenia que treballar amb llenguatge xml quan el classificador s'havia generat en una estructura de directoris i fitxers. Per solucionar aquest problema es va utilitzar una aplicació feta en c++ anomenada *haarconv*. Un cop ja es tenia l'estructura en format xml es va procedir a escriure l'aplicació *facedetect* que es l'encarregada de verificar el correcte funcionament del detector.

2.4.2. Haarconv

La càrrega d'una cascada de classificació en versions anteriors d'Open CV de Intel es realitzava mitjançant el prototip cvLoadHaarClassifierCascade però aquesta actualment ja es considerada obsoleta per la mateixa Intel. Des de la documentació de la llibreria és va veure que actualment les dades se emmagatzemen en format xml. Per realitzar la conversió, es càrrega la cascada des de directori i mitjançant la funció cvSave es grava ja en format xml. Per carregar les dades només s'ha de fer servir la funció cvLoad però tenint en compte la conversió a l'estructura correcta, en aquest cas: CvHaarClassifierCascade.

La crida tipus que s'ha fet servir és la següent:

haarconv –path c:\classificador -name c:\classificador.xml –w 24 –h 24

-path: assenyala on està emmagatzemat el classificador.
-name: nom que vulguem que tingui el classificador amb nou format sense oblidar l'extensió .xml.
-w: amplada de les mostres d'entrenament.
-h: alçada de les mostres d'entrenament.

Aquestes dues últimes opcions s'introdueixen ja que amb la instrucció cvLoadHaarClassifierCascade es necessari assenyalar la grandària de les mostres d'entrenament.

2.4.3. Procés de detecció

Per realitzar la detecció es va realitzar una aplicació anomenada *facedetect*, on tant es podia detectar en una imatge, un conjunt de imatges (mitjançant un fitxer de text), un vídeo en format avi o des de captura directa de camera.

Evidentment s'han de tenir instal·lats el conjunt de codecs necessaris per poder visualitzar els vídeos correctament.

La funció principal de l'aplicació és el prototip dissenyat per Intel cvHaarDetectObjects que es troba en el fitxer cvhaar.cpp inclòs en la llibreria cv. S'encarrega de tot el processat de la imatge o frame i proporciona un cop finalitzada la tasca de detecció, un vector amb totes les coordenades de les cares detectades.

En la crida d'aquesta funció hi trobem:

-imatge o frame

-la cascada de classificació carregada prèviament en format xml amb cvLoad a l'estructura CvHaarClassifierCascade.

-l'estructura de memòria on s'emmagatzemen les coordenades de les cares abans de finalitzar el processat i fer la transformació en un vector.

-Factor d'escala: és el factor amb el qual s'augmenta la finestra de cerca entre escombrats subseqüents. Per exemple, 1.1 incrementa la finestra un 10%.

-Veïns mínims: mínim número (menys 1) de rectangles veïns que emmascaren un objecte, en aquest cas cares. Tots els grups de rectangles de número més petit que veïns mínims menys 1 són rebutjats. Si veïns mínims es igual a 0, tots els rectangles candidats de contenir l'objecte seran retornats. Aquesta opció és per si es realitza un processat diferent al proposat per l'agrupació de rectangles.

S'ha de tenir molt de compte amb l'elecció del número de veïns mínims adient ja que podem reduir la taxa de detecció. Es pot donar el cas que es rebutgessin tots el rectangles fent que no quedes representada la detecció de l'objecte.

-Mode d'operació: aquest és un paràmetre que s'activa amb: la definició CV_HARR_CANNY_PRUNING (de valor 1). Quan es actiu, s'utilitza el detector Canny Edge per descartar algunes regions de la imatge que contenen molt poques o moltes vores (edge) i no contenen cares (en aquest cas). El llindar

utilitzat per aquest detector està ja afinat per detecció de cares per lo tant accelera el processat.

-Grandària mínima: es la grandària base des de la que es comença la detecció. És a dir, la cara més petita que pot detectar en la imatge.

En el processat mitjançant la funció cvHaarDetectObjects les funcions principals són:

- cvSetImagesForHaarClassifierCascade
- cvRunHaarClassifierCascade.

La primera s'encarrega de proporcionar les imatges ja normalitzades i amb la imatge integral realitzada a la segona que es la que activa la cascada de classificació per avaluar si en la finestra proporcionada es troba l'objecte candidat o no.

Al revisar la documentació de la funció cvSetImagesForHaarClassifierCascade, es va veure que a part dels paràmetres d'entrada: cascade, sqsum, sum i tiltedsum, es podia indicar un factor d'escala de la finestra de detecció.

Si valia 1 es mantenia la grandària original de la finestra (en aquest estudi de 24x24 píxels) i si es doblava, es passava a detectar cares de 48x48 augmentant el processat un factor 4 aproximadament però sense poder detectar cares més petites que 48x48 píxels.

2.4.4. Opcions de facedetect

-cascade: el path i el nom del classificador, sense oblidar l'extensió .xml
-i: nom del vídeo, nom de la imatge, fitxer de text o l'índex de la camera. Tant com en el vídeo com en la imatge, s'ha de posar el nom sencer i l'extensió. Si es vol utilitzar una col·lecció de imatges només s'han de posar els noms i paths de les imatges en un fitxer de text. Per passar d'una imatge a la següent, es tindrà que prémer una tecla. Si es vol utilitzar una camera només es tindrà que posar l'índex de la que es vulgui utilitzar (si es que hi ha més d'una) o no cal posar res si només hi ha activa una única camera.
-n: número de veïns mínim, per defecte 0 per així poder comprovar primerament que no eliminem detecció, malgrat que una bona elecció estàndard podria ser 2.
-m: mode d'operació, per defecte 0, només cal posar l'etiqueta, no cal afegir darrere paràmetre.
-vis: visualitzar vídeo, per defecte 1, només cal posar l'etiqueta, no cal afegir darrere paràmetre.
-w: amplada de la finestra d'entrenament, per defecte 24.
-h: alçada de la finestra d'entrenament, per defecte 24.
-s: factor d'escala per escombrats subseqüents, per defecte 1.25.
-ss: factor d'escala de la finestra de detecció, per defecte 1. 5.

L'elecció d'aquests dos darrers valors, es basa amb l'estudi fet per P. Viola i M. Jones que van presentar els seus resultats fent servir aquesta configuració.

En qualsevol moment de l'execució del procés de detecció per un vídeo o captura directa de camera podem finalitzar l'aplicació al prémer qualsevol tecla.

CAPÍTOL 3. ANÀLISI DE RESULTATS

3.1. Anàlisi de la configuració per defecte de haartraining

Vistes totes les possibilitats a l'hora de fer l'entrenament del sistema: taxa de falsa alarma, taxa de detecció, arquitectura en arbre o cascada, número de branques, número de mostres i d'altres paràmetres, es va considerar necessari fer un primer anàlisi dels paràmetres per defecte de l'aplicació de Intel. L'anàlisi i estudi d'aquesta configuració servirà per definir unes primeres pautes de treball que es milloraran posteriorment al fer un estudi detallat.

El primer pas va ser utilitzar el classificador, ja proporcionat per Intel, anomenat: "haarcascade_frontalface_default.xml", format per 24 etapes on realment no es podia saber quines taxes de falsa alarma i de detecció s'havien utilitzat. Es va utilitzar amb imatges, no amb seqüències de imatges i es va comprovar que el temps de processat era altíssim per cadascuna de les imatges utilitzades malgrat obtenir bons resultats visuals en la detecció però altes taxes de falsa alarma.

En aquest punt es va decidir veure quines eren les opcions per defecte que oferia l'aplicació *haartraining* per la creació d'un classificador. Com paràmetres més importants teníem:

- Número de mostres positives: 2000
- Número de mostres negatives: 2000
- Número d'etapes: 14
- Màxima falsa alarma per etapa: 0.5
- Mínima taxa de detecció: 0.995
- Col·lecció de característiques: BASIC
- Número de splits (branques): 1

Amb aquesta configuració s'obtindria un arbre de classificació d'una branca (equivalent a una cascada de classificadors) amb una taxa de falsa alarma global de $61 \cdot 10^{-6}$ ($0.5^{\wedge 14}$) i una taxa de detecció global de 93.2 ($0.995^{\wedge 14}$).

Es va realitzar un classificador amb aquesta configuració i els resultats obtinguts no van ser els esperats, a part de la lentitud del procés de detecció, que s'explicarà posteriorment, hi havia moltes cares que no eren detectades i les falses alarmes eren altes.

Aquest fet va fer pensar que el problema venia donat en l'elecció del número i tipus de mostres utilitzades.

A l'inici de l'estudi partíem d'una col·lecció d'unes 5000 mostres positives i de unes 3500 imatges de fons aproximadament (notar que l'algoritme d'entrenament extreu de les imatges de fons les mostres negatives de 24x24 píxels) i l'elecció inicial dels 2000 exemples negatius i els 2000 exemples positius va ésser totalment aleatòria.

En aquest punt es van revisar totes les imatges tan positives com negatives i es va poder extreure un parell de conclusions:

1) La col·lecció de mostres positives era massa variada en la posició, orientació/rotació de les cares i a més l'alineació de les línies dels ulls no eren homogènies en tot el conjunt de les imatges. És a dir, a vegades la línia dels ulls quedava desplaçada amunt o avall. Això era degut a que de la manera que es va realitzar la preparació de les mostres positives, si la cara estava una mica girada i amb una certa rotació vertical, la línia dels ulls quedava desplaçada amunt o avall en funció de l'angle de rotació. Aquest fet era un problema ja que el mètode de Boosting es basa en trobar les millors característiques que defineixen l'objecte a detectar i s'estava perdent la potència del mètode ja que es perdia homogeneïtat en el conjunt de mostres positives. A més, al revisar la documentació, una recomanació era que la situació de les línies dels ulls de tots els exemples positius fossin los més semblades possibles, que estiguessin lo més alineades possible. Es van suprimir totes les mostres on les cares estiguessin girades un cert grau i rotades verticalment. També es va aprofitar per eliminar les mostres que tenien una qualitat molt baixa. Després d'aquesta tria es va acabar amb unes 3325 mostres positives.

2) També es van revisar totes les imatges de fons i es va notar que moltes eren redundants, contenien informació molt semblada o molt repetida, fent que a l'algoritme d'entrenament l'hi fos difícil d'extreure informació útil per l'entrenament passades les primeres etapes. Es van eliminar les sobrants i es va a tornar a cercar més imatges on sobretot hi hagués més variacions dintre de la mateixa imatge i també cercant imatges on hi hagués informació que pogués tenir semblança amb les característiques de l'objecte a detectar.

Es va a tornar a provar la mateixa configuració i els resultats no van ser concloents, malgrat que les falses alarmes disminuïen i la detecció augmentava. A continuació es detallarà com s'ha anat creant un protocol de l'elecció dels diferents paràmetres fins arribar a obtenir un detector de cares ràpid i fiable.

3.2. Elecció del número d'exemples positius i negatius

Encara que els resultats eren relativament acceptables, encara calia establir una relació entre el número de mostres òptim per obtenir els resultats assenyalats teòricament.

Es va revisar l'estudi realitzat per Raquel Úbeda de La universitat Politècnica de València, veure [7], on en les seves conclusions assenyalava una manera d'escollir el número de imatges de fons òptim. En un hipotètic entrenament on volem que es trobin 3000 candidats negatius amb un número d'etapes concrets tindrem:

$$n_{imatges} = \frac{3000 \cdot n_{etapes}}{n_{candidats/imatge}} \quad \textbf{(4.1)}$$

on el número de candidats per imatge seria:

$$n_{candidats/imatge} = \frac{area_{imatge}}{area_{objecte}} \quad \textbf{(4.2)}$$

Si les imatges fossin de 384x288 píxels, grandària d'entrenament de 20x20, amb 14 etapes tindríem prou amb unes 150 imatges més o menys. Va assenyalar que després de nombroses proves realitzades, l'algoritme d'entrenament rebutjava molts candidats extrets de les imatges ja que no proporcionaven informació nova, arribant a la conclusió pràctica que s'han de proporcionar tantes imatges de fons com a candidats es requereixi trobar com a mínim. Malgrat que era una opció ja utilitzada per Viola i Jones que feien servir aproximadament el doble de candidats negatius que de positius. Si seguíssim aquesta opció tindríem que tenir 6650 imatges de fons per tenir disponibles segur 6650 candidats negatius no quedant clar que realment s'arribessin a utilitzar totes les imatges per extreure candidats negatius (recordar que l'algoritme d'extracció de candidats negatius va de imatge en imatge).

Es va intentar arribar a un compromís entre la proposta de P. Viola i M. Jones i l'aplicació realitzada per l'equip de R. Lienhart, que realment és la que s'ha utilitzat en aquest estudi. Es va pensar en un terme mitjà, ni el doble ni la unitat, però fent proves es va veure que disminuint el número de candidats negatius el resultat continuava mantenint la robustes i fiabilitat, i també es rebaixa el temps d'entrenament, hi havia prou amb que es complís:

$$n_{candidats\_negatius} \geq \frac{4}{3} n_{candidats\_positius} \quad \textbf{(4.3)}$$

Com teníem 3325 candidats positius, es van fer servir uns 4500 candidats negatius per etapa aproximadament.

L'altre problema ja anomenat abans era el fet que els temps de detecció era molt alt fent que aquesta configuració no fos l'adient per sistemes de detecció en temps real, on la rapidesa de la detecció és clau per possibles processats posteriors (extracció de característiques facials, reconeixement , autentificació).

Això es produïa pel fet que la taxa de falsa alarma era de 0.5 i aleshores un gran número de subfinestres on realment no es trobés l'objecte a detectar passaven moltes etapes prèvies. L'eficàcia de l'estructura en cascada quedava

anul·lada ja que en una imatge hi ha un número aclaparador de subfinestres que no contenen cares.

P. Viola i M. Jones en el seu estudi van assenyalar que la 1ª etapa tenia una taxa de detecció del 100% i una taxa de falsa alarma del 40% (es rebutjaven el 60%) amb només dues característiques utilitzades en l'etapa, i la segona formada per 5 característiques, rebutjava el 80% de candidats negatius amb una taxa de detecció del 100%.

Aquest punt era crític i mereixia una atenció especial, s'havien de provar diferents configuracions (cascada de classificadors o arbre de detecció) tenint en compte la relació de les taxes de falsa alarma i de detecció amb el número de característiques emprades per cadascuna de les etapes.

3.3. Cascades de classificadors vs arbres de detecció

Com ja s'ha vist en l'apartat anterior, obtenir unes etapes inicials amb una bona relació entre les taxes de falsa alarma i de detecció és un factor molt important per obtenir un bon classificador per aplicacions en temps real. Sense oblidar, que el número de les característiques emprades en aquestes etapes també és un factor clau.

Com l'aplicació utilitza una taxa màxima de falsa alarma i una taxa mínima de detecció fixa per a la construcció de totes les etapes, es va fer un estudi de només la primera etapa ja que l'elecció de les mateixes, serà emprada per la construcció de les etapes posteriors.

També es va realitzar aquest estudi amb classificadors d'una, dos i tres branques per poder saber quin del tres proporcionava millor rendiment.

Es va tenir en compte també l'ús de les col·leccions de característiques bàsica (BASIC) o complerta (ALL), veure punt 2.3.4. Opcions de Haartraining. No es va utilitzar l'opció CORE ja que en relació amb l'opció BASIC només afegia la característica point, i en canvi en l'opció ALL si es contemplava l'ús d'aquesta característica a part d'altres de no s'utilitzaven en l'opció BASIC.

S'han utilitzat dues taxes de falsa alarma màxima i dues de detecció mínima. Per escollir les taxes de detecció mínimes, es va decidir agafar un valor per damunt (0.998) de l'opció per defecte de l'aplicació (0.995) i un valor per sota (0.99). Com valors de falsa alarma màxima es van agafar de 0.2 i 0.3.

A continuació es poden veure les taules per cadascuna de les configuracions emprades, la taula 3.1. corresponen a l'estudi amb configuració d'arbre de detecció d'una branca (equivalent a cascada de classificadors, veure figura 1.5.), la taula 3.2. corresponen a dues branques i la taula 3.3. corresponen a tres branques (veure figura 1.6.):

Taula 3.1. Configuracions per un split o branca.

	HitRate	FalseAlarm	Nº Carac.	MinHitRate	MaxFalseAlarm	Mode
1	0,99825	0,2955	9	0,998	0,3	B
2	0,99825	0,2875	8	0,998	0,3	A
3	0,99825	0,1845	17	0,998	0,2	B
4	0,99825	0,188	17	0,998	0,2	A
5	0,991	0,243	6	0,99	0,3	B
6	0,99075	0,23225	5	0,99	0,3	A
7	0,99075	0,19525	7	0,99	0,2	B
8	0,9905	0,199	6	0,99	0,2	A

Taula 3.2. Configuracions per dos splits o branques.

	HitRate	FalseAlarm	Nº Carac.	MinHitRate	MaxFalseAlarm	Mode
1	0,99825	0,23175	14	0,998	0,3	B
2	0,99825	0,28375	12	0,998	0,3	A
3	0,99825	0,15950	22	0,998	0,2	B
4	0,99825	0,15875	18	0,998	0,2	A
5	0,992	0,21525	6	0,99	0,3	B
6	0,99175	0,20075	6	0,99	0,3	A
7	0,99175	0,16550	10	0,99	0,2	B
8	0,991	0,17750	8	0,99	0,2	A

Taula 3.3. Configuracions per tres splits o branques.

	HitRate	FalseAlarm	Nº Carac.	MinHitRate	MaxFalseAlarm	Mode
1	0,9983	0,2570	15	0,998	0,3	B
2	0,9983	0,2755	12	0,998	0,3	A
3	0,9983	0,1550	21	0,998	0,2	B
4	0,9983	0,1825	21	0,998	0,2	A
5	0,9913	0,2548	9	0,99	0,3	B
6	0,9943	0,1918	9	0,99	0,3	A
7	0,9905	0,1915	12	0,99	0,2	B
8	0,9943	0,1918	9	0,99	0,2	A

Al observar els resultats el primer que es va pensar en utilitzar la configuració 6 del classificador amb 2 splits, però el fet que tingués un característica més que la configuració 6 del classificador de 1 split fa fer que s'utilitzes aquesta darrera opció com a prova, ja que la carrega computacional seria menor al usar una

característica menys. Al fer servir aquesta opció tindríem que en les dues primeres etapes tindríem una taxa de falsa alarma global de 0.09 ($0.3^{\wedge 2}$) quan Viola i Jones la varen obtenir de 0.08 (0.2*0.4), a més amb menys característiques ja en la primera etapa.

Després de realitzar l'entrenament d'aquest classificador es va veure que els resultats eren molt dolents ja que la taxa de detecció global sortia de aproximadament 0.868 ($0.99^{\wedge 14}$). Es va comprovar que el classificador que s'obtenia detectava bé les cares en les primeres etapes però a partir de la sisena etapa començava a descartar cares fet que produïa que al final de la cascada es tingues una detecció molt baixa. No es podia fer servir com a taxa de detecció de 0.998, malgrat que s'obtindria un classificador amb una taxa global de 0.972, ja que l'ús de característiques en la confecció de la primera etapa ja el molt alt fet que feia que es descartés directament.

Es va fer provar amb una configuració semblada a la 6 de només un split però es va variar la taxa mínima de detecció a la que ve per defecte en l'aplicació (0.995) per veure si els resultats obtinguts eren semblats als realitzats amb una taxa de detecció de 0.99. Notar que amb aquesta elecció és millora la taxa global de detecció a 0.932 ($0.995^{\wedge 14}$).

Realment els resultats van ser molt bons, es mantenia el número de característiques emprades i les taxes obtingudes en la primera etapa amb la configuració 6 del classificador de només un split.

Es va realitzar la mateixa prova de veure etapa a etapa la evolució de les deteccions i de les falses alarmes i es va notar que la majoria de candidats negatius quedaven descartats en les 5 primeres etapes i que després amb unes 5-6 etapes més quedaven descartades totes les falses alarmes i es mantenien les deteccions correctament. Aquest fet va fer que es decidir només utilitzar 11. S'obtenien millors resultats en la taxa de detecció, 0.947 i en la taxa de falsa alarma, $1.1 \cdot 10^{-6}$.

Aquesta és configuració que s'ha fet servir per realitzar el classificador final ja que va oferir els resultats necessaris per una aplicació de vídeo en temps real. Malgrat que, hi havia uns factors molt importants utilitzats en la detecció que es tenien que ajustar per obtenir bons temps de processat en funció de la màquina emprada per aquesta finalitat. Aquests factors són el factor d'escala i la grandària mínima que es veuran en el següent apartat.

3.4. Factor d'escala i grandària mínima

Com ja es va veure en la apartat 2.4.2. Procés de detecció, hi ha dos factors claus en la velocitat de processat en la detecció. Aquest són: el factor d'escala (step size) i la grandària mínima (starting scale). El primer incrementa la finestra d'escombrat en escombrats subseqüents i el segon determina quina és el grandària mínima de l'objecte que es podrà detectar.

S'ha de pensar que aquests factors poden disminuir la taxa de detecció, i augmentar la taxa de falsa alarma que s'havien obtingut en l'entrenament del classificador.

L'opció que mantindrà les taxes obtingudes serà usant un step size de 1.1 (o el valor més proper a 1 possible) i un starting scale de 1. Amb aquesta configuració fem escombrats subseqüents amb augments del 10% de la grandària de la finestra d'escombrat i la grandària mínima serà exactament igual a la grandària de les mostres utilitzades en el procés d'entrenament, en el nostre cas 24x24.

P. Viola i M. Jones van utilitzar altres configuracions per aquests factors, els seus resultats finals es van basar en un step size de 1.5 (augmentant un 50% la finestra d'escombrat) i un starting scale de 1.25 on només podien detectar cares de 30x30 píxels (ells també van utilitzar mostres de 24x24 píxels). La maquina que van utilitzar va ésser un Pentium ® III a 700 MHz obtenint processats de 15 fps en seqüències de imatges de grandària 384x288 píxels.
En canvi, R. Lienhart va utilitzar un step size de 1.2 i un starting scale de 1, amb un Pentium ® 4 a 2 GHz aconseguint processar 5 fps en seqüències de imatges CIF.

Evidentment s'ha d'arribar a un equilibri entre la precisió i la velocitat del processat, aquest vindrà condicionat sobretot per la màquina utilitzada, per les característiques de la seqüència de imatges i sobretot de tipus d'aplicació en la que s'utilitzi el procés de detecció. Si en les imatges les cares a detectar són molt petites però estan per damunt de la grandària mínima, es tindrà que tenir molt de compte a l'hora d'escollir el factor starting scale. A més, per tenir una bona taxa de detecció s'haurà de tenir un step size baix però sense oblidar que es processaran un número major de subfinestres per imatges fet que provocarà un augment considerable de falses alarmes i del temps de processat en la detecció.

3.5. Resultats finals

Després de les proves realitzades i en funció del rendiment dels classificadors entrenats, és va decidir realitzar un entrenament d'una cascada de classificadors de 11 etapes, amb una taxa mínima de detecció de 0.995 i una taxa de falsa alarma de 0.3 per cadascuna de les etapes, 3325 mostres positives i 4500 mostres negatives. Es va obtenir un classificador amb una taxa de detecció del 94.7% i d'una taxa de falsa alarma del 0.00011%. El número de característiques per etapa es pot veure en la taula següent:

Taula 3.4. Número de característiques per etapa

Etapa	1	2	3	4	5	6	7	8	9	10	11
Nº de característiques	6	10	12	26	18	19	23	28	29	32	33

Respecte a la velocitat de processat, es van fer proves amb 3 màquines o equips diferents fins a arribar a la configuració necessària per poder processar seqüències de imatges CIF a 15fps:

1) Pentium ® 4, 2.45 GHz amb 768 MB de memòria RAM, veure taula 3.5.

Taula 3.5. Resultat obtinguts amb l'equip 1

Step Size	Starting Scale	Temps de processat frame [s]	fps
1,1	1	0,097	10,3
1,15	1	0,078	12,8
1,2	1	0,063	15,9

2) Pentium ® 4, 1.7 GHz amb 256 MB de memòria RAM, equip portàtil, veure taula 3.6.

Taula 3.6. Resultat obtinguts amb l'equip 2

Step Size	Starting Scale	Temps de processat frame [s]	fps
1,1	1	0,150	6,7
1,15	1	0,110	9,1
1,2	1	0,090	11,1
1,25	1	0,080	12,5
1,3	1	0,071	14,1
1,35	1	0,070	14,3
1,4	1	0,064	15,6

3) Athlon ® XP, 1.45 GHz amb 768 MB de memòria RAM, veure taula 3.6.

Taula 3.6. Resultat obtinguts amb l'equip 3

Step Size	Starting Scale	Temps de processat frame [s]	fps
1,1	1	0,150	6,7
1,15	1	0,110	9,1
1,2	1	0,090	11,1
1,25	1	0,080	12,5
1,3	1	0,072	13,9
1,35	1	0,070	14,3
1,4	1	0,065	15,4

Els resultats dels equips 1 i 2 són gairebé idèntics, es pot veure que una velocitat menor del processador pot ser compensar amb més memòria RAM.

Els vídeos utilitzats per les proves eren de 15 fps, grandària CIF, sense compressió i amb només una cara frontal, amb petites variacions de posició i rotació, en tota la seqüència de imatges.

CONCLUSIONS

Quan es va començar aquest estudi per realitzar un detector de cares automàtic, en cap moment sabíem del gran potencial del mètode de Boosting, i més concretament de l'AdaBoost.

La seva força, pensem, resideix en l'ús tan eficaç de conceptes tan vertaderament simples per arribar a obtenir estructures tan eficients. De com mitjançant operacions tan senzilles es podia arribar a un resultat tan robust i fiable.

De això, el fet de que tant s'hagi investigat posteriorment amb aquest mètode per millorar les seves prestacions en sistemes de detecció d'objectes en temps real. Des de Y. Freund y R.E.Schapire, fins R. Lienhart, passant per P. Viola y Michael Jones, sempre ha sigut una optimització del mètode oferint actualment una gran eina per obtenir sistemes de detecció ràpids, fiables i robustos molt apropiats per sistemes de detecció en temps real.

En el nostre cas, no ha estat una tasca de millora, si no que més bé ha estat en com utilitzar aquestes eines disponibles per obtenir un sistema de detecció de cares eficient.

La nostre filosofia en aquest estudi ha sigut intentar obtenir unes pautes o criteris d'ús per poder escollir la arquitectura o configuració més adient en funció de les premisses inicials que eren obtenir un sistema de detecció automàtic en sistemes de vídeo en temps real amb una taxa de detecció alta i d'una taxa de falsa alarma baixa.

Per les característiques de l'aplicació d'entrenament feta servir, s'han tingut que realitzar moltes proves per observar els resultats que s'anaven obtenint en funció dels paràmetres utilitzats i així de mica en mica poder arribar a realitzar una certa generalització de l'ús de la aplicació i de com utilitzar els paràmetres disponibles adequadament per obtenir al final un resultat satisfactori.

Ha estat un cicle de treball realimentat, és a dir, primer es realitzaven proves d'entrenament per obtenir un classificador, que no sempre era del tot acceptable però que ens servien per evolucionar la manera de treballar amb els paràmetres adequadament mitjançant l'estudi del classificador realitzat. S'estudiaven per així tornar a començar el cicle fins acotar l'ús i poder tenir al final un classificador adient per la nostre aplicació.

A partir de la configuració per defecte de l'algoritme d'entrenament i del posterior estudi realitzat, es va arribar a un protocol d'elecció del paràmetres disponibles per obtenir un detector per a ús en temps reals.

1) L'elecció del número de mostres per realitzar el procés d'entrenament d'una manera òptima, vindrà condicionat pel fet que el número de mostres negatives han d'ésser com a mínim igual a 4/3 del número de mostres negatives. On les mostres

positives deuen ser lo més homogènies possible dins de la màxima variabilitat possible, és a dir, en el cas de l'ús de cares, utilitzar només cares frontals sense excessiva rotació ni horitzontal ni vertical. Respecte a les mostres negatives, que siguin lo més variades possible evitant usar imatges que continguin informació molt similar i si pot ser, que continguin informació susceptible de ser semblada a una cara. En aquest estudi es va reduir el número d'exemples positius i negatius necessaris per el proces d'entrenament d'un detector de cares en temps real respecte altres treballs anteriors. Aquest fet redueix el temps d'entrenament. S'han usat 3325 exemple positius i 4500 exemples negatius.

2) L'ús de cascades de classificadors en front a arbres de detecció ja que per el simple fet d'usar menys característiques simples per etapa computacionalment la primera opció serà més eficient.

3) Reduir la taxa de falsa alarma mantenint una taxa de detecció molt alta en funció del número de característiques usades seguint la línia proposada per P. Viola i M. Jones. Es va establir una taxa de falsa alarma màxima de 0.3 i una taxa de detecció mínima de 0.995. Aquesta elecció a part de donar molts bons resultats globalment, afavoria la reducció del número total d'etapes utilitzades en la confecció del classificador o detector. Amb l'ús de 6 característiques simples en la primera etapa i 10 en la segona etapa, sent les dues les etapes més critiques d'una cascada de classificadors en les quals es pot reduir notablement el temps de processat en la detecció.

4) L'ús de la col·lecció complerta de característiques, les mateixes que va usar R. Lienhart, ja que s'ha comprovat que amb l'ús de mostres positives de cares exclusivament frontals i sense una rotació excessiva, posteriorment, en el procés de detecció s'amplia l'espectre de cares que es poden detectar arribant a inclòs a detectar cares que en un principi semblaven ser susceptibles de no ser detectades. Aquest fet es degut al l'ús de les característiques rotades però té l'inconvenient que augmentar el temps d'entrenament ja que el número de característiques a avaluar es major i a més les característiques rotades tenen una càrrega computacional major. Malgrat que aquest inconvenient queda ràpidament compensat amb el fet d'obtenir una taxa de detecció molt millor que si només utilitzéssim característiques verticals.

5) Flexibilitat de certs paràmetres en el procés de detecció com l'elecció de: step size, starting scale i número de veins. Amb aquesta opció s'obté un classificador adaptable, no estàtic, amb un cert punt de configuració per part de l'usuari que utilitzi el detector. A més es podrà adequar a l'aplicació en la que s'hagi d'integrar o simplement adaptar-lo a la màquina amb la que es

treballi. Malgrat que s'ha de tenir en compte ja que una configuració errònia d'aquest paràmetres portaran a una degradació de les taxes de detecció i falsa alarma.

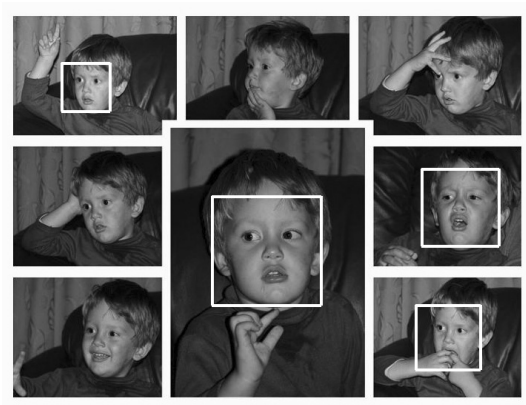
Malgrat que es podien haver afegit certes millores, cosa molt difícil de realitzar per la complexitat de l'aplicació d'entrenament, tal com: l'opció d'escollir les taxes, de detecció mínima i de falsa alarma màxima, independentment per cadascuna de les etapes i l'opció d'escollir el número màxim de característiques simples emprades en cadascuna de les etapes.

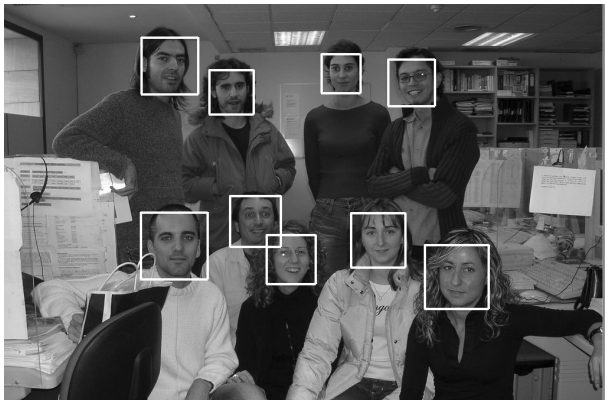
En definitiva, en aquest treball, s'ha intentat explicar com arribar a un compromís adequat entre la càrrega computacional del detector i la velocitat de processat sense disminuir la taxa de detecció ni augmentar la taxa de falsa alarma per obtenir un sistema fiable, robust i ràpid on la taxa de detecció es alta i la taxa de falsa alarma baixa.

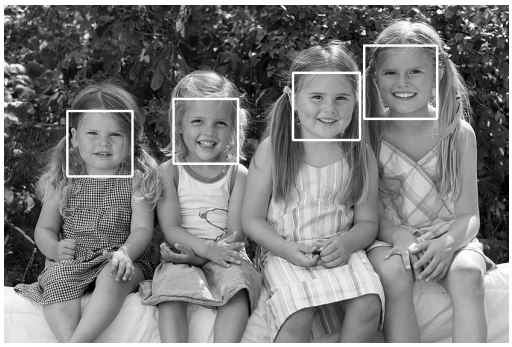
En el nostre cas s'ha realitzat detector de cares automàtic amb una cascada de classificadors de 11 etapes, una taxa de detecció de 94.7% i una taxa de falsa alarma de 0.00011%, el qual pot processar seqüències de imatges CIF de 15 fps.

ANNEXES

S'han volgut afegir en els annexes una sèrie de imatges on es pot comprovar visualment els resultats obtinguts amb el classificador realitzat. S'ha utilitzat en el procés de detecció un step size de 1.1 i un starting scale de 1. Es pot apreciar la baixa taxa de falsa alarma i l'alta taxa de detecció, malgrat que hi ha imatges on hi ha cares amb una rotació força gran o cares molt petites que no són detectades.







BIBLIOGRAFIA

[1] Freund, Y. and Schapire, R. E. “A Descision-teoretic Generalization of On-line Learning and an Application to Boosting”. In *Second European Conference on Computational Learning Theory*, 1995.

[2] Papageorgiou, Oren, M. and Poggio, T. “A General Framework for Object Detection”. In *International Conference on Computer Vision*. 1998.

[3] Viola, P. and Jones, M. “Robust real-time object detection”. In *Second International Workshop on Statical and Computational Theories of Vision-Modeling, Learning, Computing, and Sampling*. 2001.

[4] Viola, P. and Jones, M. “Rapid Object Detection Using a Boosted Cascade of Simple Features”. In *Accepted Conference on Computer Vision and Pattern Recognition*. 2001.

[5] Lienhart, R., Liang, L. and Kuranov, A. “A Detector Tree of Boosted Classifiers for Real-Time Object Detection and Tracking”. *Technical report*, Intel Corporation. 2002.

[6] Lienhart, R. and Maydt, J. “An Extended Set of Haar-like Features for Rapid Object Detection”. *Technical report*, Intel Corporation. 2002.

[7] Úbeda, Raquel. “Aplicación de Técnicas de Boosting para Detección de Ma - trículas”. *PFC*. Universidad Politécnica de Valencia. 2004..